# Calcolo Numerico

Federico Matteoni

A.A. 2019/20

# Indice

# Introduzione

Prof.: Luca Gemignani

**Calcolo Numerico** Metodi numerici per risolvere problemi matematici con il calcolatore. In questo corso ce ne occuperemo dal punto di vista numerico, perché metodi di risoluzione diversi performano in maniera diversa sulla macchina. Cerchiamo di capire quali sono i metodi di interesse e cosa aspettarci dalla loro implementazione.
Il **metodi numerici approssimano la soluzione di problemi matematici**.
Inoltre, il computer **impatta** sul calcolo perché lavora con approssimazioni dei numeri, che su moli elevate di dati e di elaborazioni finiscono per perturbare il risultato ottenuto.

**Tipici problemi**

$$Ax = b$$

$$Ax = \lambda x$$

$$f(x) = 0$$

$$\int_a^b f(x)dx$$

**Matlab** Matrix Laboratory, strumento di implementazione per verificare e constatare i risultati teorici.

**Informazioni d'esame** Compitini, che se complessivamente passati rendono l'orale facoltativo. In alternativa, appelli scritti + orale.

# Capitolo 1

# Aritmetica di macchina

Modello per capire cosa aspettarci dal punto di vista degli errori dell'esecuzione.

**Esempio** Per calcolare il limite

$$\lim_{x\to\infty} \sqrt{x+1} - \sqrt{x}$$

ottengo un caso indeterminato $(\infty - \infty)$. Posso semplificare l'espressione ad esempio razionalizzando, e con pochi passaggi ottengo la seguente uguaglianza

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

Quindi dal punto di vista matematico, le due espressioni sono equivalenti. Ciò **non è sempre vero per la rappresentazione ed il calcolo in macchina**: le due espressioni forniranno risultati completamente diversi. Si rende **necessario**, quindi, **capire quale metodo si comporta meglio** rispetto agli altri.

## 1.1 Rappresentazione in macchina

**Rappresentare i numeri** Siamo comunemente abituati a rappresentare un numero in diverse forme.
Ad esempio, $0.1 = \frac{1}{10} = 10 \cdot 0.01$. In generale, per ogni numero ho diversi metodi di rappresentazione $\Rightarrow$ In macchina dobbiamo poter **rappresentare i numeri in maniera univoca**.

**Base di numerazione** $B \in N, B > 1$, poiché in base 1 non si può contare.
Una base $B$ ha cifre nell'insieme $\{0, 1, \ldots, B-1\}$

**Teorema** Dato $x \in R$, con $x \neq 0$
$\exists!$ $(\{d_i\}_{i\geq 1}$ con $d_1 \neq 0$ e $d_i$ non definitivamente uguali a $B-1) \wedge (p \in Z) \mid x = segno(x) \cdot B^p \cdot \sum_{i=1}^{\infty} d_i \cdot B^{-i}$
Considerazioni

Se $x \in C$ allora viene rappresentato come coppia di numeri reali, quindi il problema si riconduce sempre alla loro rappresentazione

Lo 0 viene rappresentato in modo speciale, poiché non esiste una sua rappresentazione normalizzata

$d_{ii\geq 1}$ è una **successione** di cifre

La rappresentazione è **normalizzata** se $d_1 \neq 0$, cioè se la prima cifra è diversa da 0

Non può avere tutte cifre uguali a $B-1$ da un certo punto in poi, la rappresentazione "collassa" al numero successivo

$p$ è detto **esponente**

$\sum_{i=1}^{\infty} d_i \cdot B^{-i}$ è detta **mantissa**

Questa rappresentazione si chiama **in virgola mobile** o **floating point**

| Segno | Esponente | Mantissa |
|---|---|---|

**Esempi** Ponendo $B = 10$

$x = 0.01 \Rightarrow x = +10^{-1} \cdot (0.1)$

$x = 1.35 \Rightarrow x = +10^{1} \cdot (0.135)$

$x = 0.0023 \Rightarrow x = +10^{-2} \cdot (0.23)$

**Numeri di Macchina** Nei computer ho **registri di lunghezza finita**, quindi essi vengono partizionati: una parte viene riservata alla rappresentazione dell'**esponente** e il resto alla rappresentazione della **mantissa**. L'**insieme dei numeri di macchina** F è quindi così definito

$$F(B,t,m,M) = \{\pm B^p \cdot \sum_{i=1}^{t} d_i \cdot b^{-i} \;\Big|\; d_1 \neq 0 \wedge -m \leq p \leq M\} \cup \{0\}$$

dove

$t$ sono le **cifre della mantissa**

L'esponente p è compreso tra i valori $-m$ e $M$

Lo 0 è incluso ma rappresentato a parte

**Esempio** Ipotizzando di usare registri da 32 bit, posso stanziare 8 bit per l'esponente $p$ (quindi 1 bit per il segno e 7 bit per il valore) e i restanti 24 bit per la mantissa (1 per il segno, 23 per il valore). Quanti numeri posso rappresentare?

$p$ di 7 bit $\Rightarrow 2^7 - 1 = 127 \Rightarrow -127 \leq p \leq 127$ ma lo 0 è rappresentato due volte

$x = \pm 2^p \sum_{i=1}^{2} 3d_i \cdot 2^{-1}$, con $d_i \in \{0,1\}$, $d_1 \neq 0 \Rightarrow d_1 = 1$ sempre

Vedremo che con una serie di accorgimenti è possibile aumentare i numeri esattamente rappresentabili.

Dato quindi $F(B,t,m,M)$, osservo che:

$F(B,t,m,M)$ ha cardinalità finita $N = 2B^{t-1}(B-1)(M+m+1)+1$

Se $x \in F(B,t,m,M) \wedge x \neq 0 \Rightarrow \omega = B^{-m-1} \leq |x| \leq B^M(1-B^{-t}) = \Omega$
Quindi non è possibile rappresentare esattamente numeri non nulli minori di $\omega$. Si può introdurre una rappresentazione **denormalizzata** quando $p = -m$: la condizione $d_1 \neq 0$ può essere abbandonata e posso così rappresentare numeri positivi e negativi compresi in modulo tra $B^{-m-t}$ e $B^{-m}(B^{-1} - B^{-t})$
Analogamente se $p = M$ si introducono rappresentazioni speciali per i simboli $\pm\infty$ e NaN (**not a number**).

### 1.1.1 Aritmetica di Macchina

La rappresentazione di un numero $x \in R, x \neq 0$ in macchina significa **approssimare** $x$ con un numero $\overline{x} \in F(B,t,m,M)$ commettendo un **errore relativo** di rappresentazione

$$\epsilon_x = \frac{\overline{x} - x}{x} = \frac{\eta_x}{x}, x \neq 0$$

quanto più piccolo possibile in valore assoluto. La quantità $\eta_x = \overline{x} - x$ è detta **errore assoluto** della rappresentazione. L'errore relativo è importante per la **valutazione qualitativa** dell'errore: nelle misurazioni astronomiche un errore assoluto di 1 cm è *qualitativamente diverso* da un errore assoluto di 1 cm nella misurazione di un tavolo.

Dato $x \in R, x \neq 0$, distinguiamo due casi:

1. $|x| < \omega$ (**underflow**) oppure $|x| > \Omega$ (**overflow**)
2. $\omega \leq |x| \leq \Omega$

Nel secondo caso abbiamo quattro tecniche di approssimazione:

1. **Arrotondamento**: $x$ approssimato con il numero rappresentabile $\overline{x}$ più vicino
2. **Troncamento**: $x$ approssimato con il più grande numero rappresentabile $\overline{x}$ tale che $|\overline{x}| \leq |x|$
3. *Round toward* $+\infty$: $x$ approssimato al più piccolo numero rappresentabile maggiore del dato
4. *Round toward* $-\infty$: $x$ approssimato al più grande numero rappresentabile minore del dato

Per semplicità considereremo una macchina che opera per troncamento sull'insieme $F(B,t,m,M)$.
Indicheremo con $trn(x) = \overline{x}$ il risultato dell'approssimazione di $x$ con troncamento e più in generale $fl(x)$ l'**approssimazione in macchina del dato** $x$ nel sistema floating point considerato.

**Teorema** Sia $x \in R$ con $\omega \leq |x| \leq \Omega$. Si ha

$$|\epsilon_x| = |\frac{trn(x) - x}{x}| \leq u = B^{1-t}$$

Si osserva che:

$u = B^{1-t}$ è detta **precisione di macchina** ed è **indipendente dalla grandezza del numero**, ma è caratteristica dell'aritmetica floating point, dell'insieme dei numeri rappresentabili e dalla tecnica di approssimazione. Se ad esempio si opera con arrotondamento, $u$ si dimezza.

Per valutare la precisione di macchina possiamo determinare il più piccolo numero di macchina maggiore di 1. Dato $x$ tale numero, abbiamo $x - 1 = |x - 1| = B^{1-t}$ essendo $1 = B^1 \cdot B^{-1}$ rappresentato con esponente $p = 1$. Il seguente script MatLab fornisce il valore richiesto:

```
eps = 0.5;
eps1 = eps + 1;
while(eps > 1)
        eps = 0.5 * eps;
        eps1 = eps + 1;
end
eps = 2 * eps;
```

Dal teorema si ricava che dato $x \in R$, in assenza di overflow/underflow, vale $fl(x) = x(1 + \epsilon_x)$ con $|\epsilon_x| \leq u$. Questa relazione esprime il modo in cui viene descritto generalmente il legame tra numero reale e sua rappresentazione in macchina.

### 1.1.2 Operazioni di Macchina

Per le **operazioni aritmetiche** in un sistema floating point si pone un analogo problema di approssimazione, in quanto **il risultato di un'operazione eseguita tra due numeri di macchina in generale non sarà un numero di macchina**.

**Operazioni** Indichiamo con $\oplus$, $\ominus$, $\otimes$, $\oslash$ le **operazioni aritmetiche di macchina** corrispondenti relativamente all'addizione, sottrazione, prodotto e divisione. Si richiede che le operazioni siano interne all'insieme dei numeri di macchina. Una ragionevole definizione, derivata dal teorema precedente e in assenza di overflow/underflow, risulta:

$$\forall a, b \in F(B,t,m,M), a \oplus b = fl(a+b) = (a+b)(1+\epsilon_1), |\epsilon_1| \leq u$$

con $\epsilon_1$ detto **errore locale dell'operazione**. Sempre in assenza di overflow/underflow, se $a, b \in R$ si ha

$$fl(a+b) = fl(a) \oplus fl(b) = (a(1+\epsilon_a) + b(1+\epsilon_b))(1+\epsilon_1) \doteq (a+b) + a\epsilon_a + b\epsilon_b + (a+b)\epsilon_1$$

dove con $\doteq$ si indica che l'eguaglianza vale **considerando le sole componenti lineari negli errori** e trascurando le componenti di ordine superiore (**analisi al primo ordine dell'errore**), in virtù del fatto che gli $\epsilon$ sono quantità piccole $0 < \epsilon < 1$, quindi trascurabili negli ordini superiori al primo.
Si ottiene che, in assenza di overflow/underflow, se $a, b \in R, a + b \neq 0$, allora

$$\frac{fl(a+b) - (a+b)}{a+b} \doteq \frac{a}{a+b}\epsilon_a + \frac{b}{a+b}\epsilon_b + \epsilon_1$$

che esprime la dipendenza dell'errore totale commesso nel calcolo della somma tra due numeri reali rispetto agli errori generati dall'approssimazione dei dati iniziali (**errore inerente**) e agli errori generati dall'algoritmo di calcolo (**errore algoritmico**) visto come sequenza di operazioni artimetiche.

**Esempio** Analizziamo cosa succede in macchina quando proviamo a calcolare $f(x) = x^2 - 1 = (x-1)(x+1)$. La **prima situazione di errore** sia ha sulla rappresentazione di $x$ che, **in generale**, **non è un numero di macchina**.

$$x \to \tilde{x} = x(1+\epsilon_x)$$

con $|\epsilon_x| \leq u$. Inoltre, sempre in generale, **le operazioni aritmetiche non sono operazioni di macchina**

$$f(x) = (\tilde{x} \otimes \tilde{x}) \ominus 1 = (\tilde{x} \ominus 1) \otimes (\tilde{x} \oplus 1)$$

Poniamo $g_1(x) = (\tilde{x} \otimes \tilde{x}) \ominus 1$ e $g_2(x) = (\tilde{x} \ominus 1) \otimes (\tilde{x} \oplus 1)$. La formula per l'**errore totale** dell'operazione è

$$\epsilon_{tot1} = \frac{g_1(x) - f(x)}{f(x)}$$

$$\epsilon_{tot2} = \frac{g_2(x) - f(x)}{f(x)}$$

Sviluppiamo $g_1(x)$

$$\begin{aligned} g_1(x) &= ((x(1+\epsilon_x) \cdot x(1+\epsilon_x))(1+\epsilon_1) - 1)(1+\epsilon_2) \doteq \\ &\doteq (x^2(1+2\epsilon_x)(1+\epsilon_1) - 1)(1+\epsilon_2) \doteq \\ &\doteq (x^2((1+2\epsilon_x+\epsilon_1) - 1)(1+\epsilon_2) \doteq \\ &\doteq x^2(1+2\epsilon_x+\epsilon_1+\epsilon_2) - (1+\epsilon_2) = \\ &= (x^2-1) + 2x^2\epsilon_x + x^2\epsilon_1 + (x^2-1)\epsilon_2 = g_1(x) \\ &|\epsilon_i| \leq u \end{aligned}$$

Quindi, portando al primo membro dell'uguaglianza $\epsilon_{tot1} = \frac{g_1(x)-f(x)}{f(x)}$ tutti i fattori $(x^2-1) = f(x)$ si ottiene

$$\epsilon_{tot1} = \frac{2x^2}{x^2-1}\epsilon_x + \frac{x^2}{x^2-1}\epsilon_1 + \epsilon_2$$

Si evince che **l'errore totale è la somma di due componenti**:

> **Errore inerente** o inevitabile: l'**errore di rappresentazione di** $x$, che vale 0 se $x$ è numero di macchina ed è **proprietà della funzione**. Nell'esempio, $\epsilon_{in} = \frac{2x^2}{x^2-1}\epsilon_x$.
> Se l'errore inerente è piccolo si dice che **la funzione è numericamente stabile**, viceversa è **numericamente instabile**.
>
> **Errore algoritmico**, locale all'operazione e **proprietà dell'algoritmo**. Nell'esempio, $\epsilon_{alg} = \frac{x^2}{x^2-1}\epsilon_1 + \epsilon_2$.
> Se è piccolo, si dice che l'**espressione è ben condizionata** e **poco sensibile rispetto alla perturbazione**. Viceversa, è **mal condizionata**.

Sviluppiamo ora $g_2(x)$ (gli errori $\delta_i$ sono analoghi agli $\epsilon_i$, si usa una notazione differente per evidenziare che hanno valori diversi, esseno propri del calcolo e dei valori usati in esso)

$$\begin{aligned} g_2(x) &= ((x(1+\epsilon_x) - 1)(1+\delta_1))((x(1+\epsilon_x) + 1)(1+\delta_2))(1+\delta_3) \doteq \\ &\doteq (x^2(1+\epsilon_x)^2 - 1)(1+\delta_1+\delta_2+\delta_3) \doteq \\ &\doteq (x^2(1+2\epsilon_x) - 1)(1+\delta_1+\delta_2+\delta_3) = \\ &= x^2(1+2\epsilon_x+\delta_1+\delta_2+\delta_3) - (1+\delta_1+\delta_2+\delta_3) = \\ &= (x^2-1) + 2x^2\epsilon_x + (x^2-1)(\delta_1+\delta_2+\delta_3) = g_2(x) \\ &|\epsilon_x| \leq u, |\delta_i| \leq u \end{aligned}$$

Come prima, porto gli $f(x)$ al primo membro dell'uguaglianza $\epsilon_{tot2} = \frac{g_2(x)-f(x)}{f(x)}$ e ottengo

$$\epsilon_{tot2} = \frac{2x^2}{x^2-1}\epsilon_x + \delta_1 + \delta_2 + \delta_3$$

Notiamo come

$\epsilon_{in} = \frac{2x^2}{x^2-1}\epsilon_x$, come il calcolo precedente. Infatti, ripetiamo, l'errore inerente è una proprietà della funzione e non di come essa viene calcolata

$\epsilon_{alg} = \delta_1 + \delta_2 + \delta_3$, diverso poiché abbiamo seguito un calcolo differente

Per poter comparare i due algoritmi e scegliere il migliore, analizziamo gli errori algoritmici. L'analisi **va eseguita in valore assoluto**, poiché non si ha alcuna informazione sul segno degli errori, seguendo quindi le regole di comparazione dei valori assoluti:

$|a+b| \leq |a| + |b|$

$|a \cdot b| = |a| \cdot |b|$

$$\epsilon_{alg1} = |\frac{x^2}{x^2-1}\epsilon_1 + \epsilon_2| \leq |\frac{x^2}{x^2-1}\epsilon_1| + |\epsilon_2| = |\frac{x^2}{x^2-1}| \cdot |\epsilon_1| + |\epsilon_2| \leq \frac{x^2}{|x^2-1|}u + u = (\frac{x^2}{|x^2-1|} + 1)u$$

$$\epsilon_{alg2} = |\delta_1 + \delta_2 + \delta_3| \leq |\delta_1| + |\delta_2| + |\delta_3| \leq 3u$$

In $\epsilon_{alg1}$, quindi, l'errore algoritmo è minore o uguale a $(\frac{x^2}{|x^2-1|} + 1)u$, che però diventa arbitrariamente grande quando $x$ si avvicina ad 1. Il secondo algoritmo è dunque **preferibile**, poiché l'**errore algoritmico è limitato**.

**Esempio** Calcoliamo ora l'espressione $f(x) = x^2 + 1$. Poniamo $g(x) = (\tilde{x} \otimes \tilde{x}) \oplus 1$ e sviluppiamo, ottenendo

$$g(x) \doteq x^2(1 + 2\epsilon_x + \epsilon_1 + \epsilon_2) + (1 + \epsilon_2)$$

L'errore totale è quindi

$$\epsilon_{tot} = \frac{g(x) - f(x)}{f(x)} = \frac{2x^2}{x^2+1}\epsilon_x + \frac{x^2}{x^2+1}\epsilon_1 + \epsilon_2$$

Studiamo le componenti

$$|\epsilon_{in} = |\frac{2x^2}{x^2+1}\epsilon_x| = |\frac{2x^2}{x^2+1}| \cdot |\epsilon_x| \leq \frac{2x^2}{x^2+1}u \leq 2u$$

perché $\frac{x^2}{x^2+1} \leq 1$. La funzione quindi **non è suscettibile rispetto alle perturbazioni dei dati in ingresso** ed è **ben condizionata**.

$$\epsilon_{alg} = |\frac{x^2}{x^2+1}\epsilon_1 + \epsilon_2| \leq |\frac{x^2}{x^2+1}\epsilon_1| + |\epsilon_2| = \frac{x^2}{x^2+1}|\epsilon_1| + |\epsilon_2| \leq |\epsilon_1| + |\epsilon_2| \leq 2u$$

Quindi l'algoritmo è **numericamente stabile** perché la componente dell'errore algoritmo è piccola ($\leq 2u$). Questo è il caso migliore che può capitare: **funzione ben condizionata** e **algoritmo numericamente stabile**.

Succede perché non vi è la sottrazione al numeratore. La sottrazione, in macchina, lavora usando molte delle cifre "sporche" della mantissa, cioè quelle che fanno parte del rumore e dell'errore di rappresentazione. Si chiama **errore di cancellazione**.

### 1.1.3 Errore Inerente

**Definizione** Si dice **errore inerente** o **inevitabile** generato nel calcolo di $f(x) \neq 0$ la quantità

$$\epsilon_{in} = \frac{f(\tilde{x}) - f(x)}{f(x)}$$

Osserviamo che l'errore inerente è una **proprietà della funzione**, ovvero del problema matematico, e misura la **sensibilità rispetto alla perturbazione del dato iniziale**. Quindi, è indipendente dall'algoritmo di calcolo.
Se l'errore inerente è qualitativamente elevato in valore assoluto, diciamo che **il problema matematico è mal condizionato**, altrimenti si dice che è **ben condizionato**.

### 1.1.4 Errore Algoritmico

**Definizione** Si dice **errore algoritmico** generato nel calcolo di $f(x) \neq 0$ la quantità

$$\epsilon_{alg} = \frac{g(\tilde{x}) - f(\tilde{x})}{f(\tilde{x})}$$

Osserviamo che l'errore algoritmico è **dipendente dall'algoritmo utilizzato**: la funzione $g$ dipende dall'algoritmo usato per calcolare $f(x)$. In generale, **differenti algoritmi conducono a differenti errori algoritmici**.
Se l'errore algoritmico è qualitativamente elevato in valore assoluto si dice che **l'algoritmo è numericamente instabile**, altrimenti si dice che è **numericamente stabile**.

### 1.1.5 Errore Totale

**Definizione** Si dice **errore totale** generato nel calcolo di $f(x) \neq 0$ mediante l'algoritmo specificato da $g$ la quantità

$$\epsilon_{tot} = \frac{g(\tilde{x}) - f(x)}{f(x)}$$

L'errore totale misura la **differenza relativa tra l'output atteso e l'output effettivamente calcolato**.

**Teorema** In un analisi al primo ordine, vale

$$\epsilon_{tot} = \epsilon_{in} + \epsilon_{alg}$$

**Dim**

$$\epsilon_{tot} = \frac{g(\tilde{x}) - f(x)}{f(x)} = \frac{f(\tilde{x}) - f(\tilde{x})}{f(\tilde{x})} \cdot \frac{f(\tilde{x})}{f(x)} + \frac{f(\tilde{x}) - f(x)}{f(x)} = \epsilon_{alg}(1 + \epsilon_{in}) + \epsilon_{in} \doteq \epsilon_{alg} + \epsilon_{in}$$

Viene espresso il fatto che nel calcolo di una funzione razione, in un'analisi al primo ordine, le due fonti di generazione d'errore forniscono contributi separati che possono essere analizzati indipendentemente.
L'obiettivo dell'analisi numerica è **trovare algoritmi numericamente stabili per problemi ben condizionati**.

## 1.2 Tecniche per l'Analisi degli Errori

### 1.2.1 Coefficiente di Amplificazione

Dalla seguente relazione

$$\epsilon_{in} = \frac{f(\tilde{x}) - f(x)}{f(x)} = \frac{f(\tilde{x}) - f(x)}{\tilde{x} - x} \cdot \frac{x}{f(x)} \cdot \frac{\tilde{x} - x}{x}$$

si ricava che la **differenziabilità di $f(x)$ è essenziale** per il controllo dell'errore inerente. Se assumiamo che $f(x)$ è derivabile due volte e continua in $(a, b)$, allora vale lo sviluppo di Taylor

$$f(\tilde{x}) = f(x) + f'(x)(\tilde{x} - x) + \frac{f''(\xi)}{2}(\tilde{x} - x)^2, \quad |\xi - x| \leq |\tilde{x} - x|$$

da cui si ottiene

$$\epsilon_{in} = \frac{f(\tilde{x}) - f(x)}{f(x)} \doteq \frac{f'(x)}{f(x)} x \epsilon_x = c_x \epsilon_x, \quad c_x = \frac{f'(x)}{f(x)} x$$

La quantità $c_x = c_x(f) = \frac{f'(x)}{f(x)} x$ è detta **coefficiente di amplificazione** e fornisce una **misura del condizionamento del problema**.
In generale, se $f : \Omega \to R$ è definita su un insieme aperto di $R^n$, differenziabile due volte su $\Omega$ ed il segmento di estremi $\tilde{\mathbf{x}}$ e $\mathbf{x}$ è contenuto in $\Omega$ allora vale

$$\epsilon_{in} = \frac{f(\tilde{\mathbf{x}}) - f(\mathbf{x})}{f(\mathbf{x})} \doteq \frac{1}{f(\mathbf{x})} \sum_{i=1}^{n} \frac{\partial f}{\partial x_i}(\mathbf{x}) x_i \epsilon_{x_i} = \sum_{i=1}^{n} c_{x_i}(f) \epsilon_{x_i}$$

con

$$c_{x_i}(f) = \frac{1}{f(\mathbf{x})} \frac{\partial f}{\partial x_i}(\mathbf{x}) x_i, \quad 1 \leq i \leq n$$

detti **coefficienti di amplificazione della funzione $f$ rispetto alla variabile $x_i$**.

**Esempio** per $f(x) = (x^2 + 1)/x$ si ha

$$c_x = \frac{2 - (x^2 + 1)}{x^2} \cdot \frac{x}{x^2 + 1} \cdot x = \frac{x^2 - 1}{x^2 + 1}$$

e poiché $|c_x| \leq 1$ il problema risulta ben condizionato.

**Coefficienti**

Per le operazioni aritmetiche si ottiene

$$f(x,y) = x + y \quad \epsilon_{in} \doteq c_x\epsilon_x + c_y\epsilon_y \quad c_x = \frac{x}{x+y} \quad c_y = \frac{y}{x+y}$$

$$f(x,y) = x - y \quad \epsilon_{in} \doteq c_x\epsilon_x + c_y\epsilon_y \quad c_x = \frac{x}{x-y} \quad c_y = -\frac{y}{x-y}$$

$$f(x,y) = x \cdot y \quad \epsilon_{in} \doteq c_x\epsilon_x + c_y\epsilon_y \quad c_x = 1 \quad c_y = 1$$

$$f(x,y) = \frac{x}{y} \quad \epsilon_{in} \doteq c_x\epsilon_x + c_y\epsilon_y \quad c_x = 1 \quad c_y = -1$$

Ne segue, come visto in precedenza, che la sottrazione di due numeri vicini tra loro è potenziale causa di elevata amplificazione degli errori relativi (**cancellazione numerica**). Si nota anche come le operazioni moltiplicative siano ben condizionate.

**Esempio** Siano $x = 0.2178 \cdot 10^2$ e $y = 0.218 \cdot 10^2$, supponendo di operare con troncamento in base B = 10 e t = 3 cifre di rappresentazione ($u = 10^{1-t} = 10^{-2}$). Si ha $\tilde{x} = 0.217 \cdot 10^2$ e $\tilde{y} = y$.
Pertanto $\tilde{x} \oplus \tilde{y} = -0.001 \cdot 10^2 = -0.1$ mentre $x - y = -0.0002 \cdot 10^2 = -0.2 \cdot 10^{-1}$ e quindi $|\epsilon_{in}| = \frac{0.8}{0.2} = 0.4$.

# Capitolo 2

# Problemi dell'Algebra Lineare Numerica

## 2.1 Norme Matriciali e Vettoriali

I principali problemi dell'algebra lineare concernono la **risoluzione di sistemi lineari** ed il **calcolo di autovalori/autovettori di matrici**. Per studiarne il condizionamento è fondamentale avere degli strumenti per valutare la distanza tra vettori e tra matrici.
La risoluzione di sistemi lineari può essere eseguita in aritmetica reale, ma gli autovalori/autovettori possono essere complessi. Servono quindi **strumenti che operino su spazi vettoriali** $F^n$ e $F^{n\times n}$, con $n \geq 1$ e $F \in \{R, C\}$

**Condizionamento su problemi reali** $f: R^n \to R$, $G_m = \frac{f(\tilde{x})-f(x)}{f(x)}$
Problema: calcolo della soluzione di un sistema lineare

**Input** $A \in R^{2\times 2}$, $b \in R^2$

**Output** $x \in R^2 \mid Ax = b$

Condizionamento: $A$, $B \to$ devono essere approssimati a valori di macchina

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \to \widehat{A} = \begin{bmatrix} \widehat{a} & \widehat{b} \\ \widehat{c} & \widehat{d} \end{bmatrix} \text{ con } \widehat{a} = a(1+\epsilon_a) \ldots$$

$$b = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \to \widehat{b} = \begin{bmatrix} \widehat{b_1} \\ \widehat{b_2} \end{bmatrix} \text{ con } \widehat{b_i} = b_i(1+\epsilon_{b_i}) \ldots$$

$$Ax = b \to \widehat{A}\widehat{x} = \widehat{b}$$

Quando $\widehat{x}$ è vicino a $x$? Per valutare la distanza fra vettori, ho bisogno delle **norme vettoriali**

### 2.1.1 Norma Vettoriale

**Definizione** $f: R^n \to R$ si dice **Norma Vettoriale** se soddisfa le seguenti proprietà

1. $f(x) \geq 0 \wedge (f(x) = 0 \Leftrightarrow x = 0)$
2. $f(\alpha x) = |\alpha| f(x) \quad \forall x \in R^n \ \forall \alpha \in R$
3. $f(x+y) \leq f(x) + f(y) \quad \forall x, y \in R^n$

**Norma Euclidea**

$$f(x) = ||x||_2 = \sqrt{\sum_{i=1}^{n} x_i^2}$$

con $x = [x_1, x_2, \ldots, x_n]$.

**Norma Infinito**

$$f(x) = ||x||_\infty = max_{i=1}^{n}|x_i|$$

**Dimostrazione** che è una norma vettoriale

1. $f(x) = max|x_i|$ è sempre $\geq 0$
   $f(x) = max|x_i| = 0 \Leftrightarrow |x_i| = 0 \ \ \forall i \Leftrightarrow x = 0$
2. $f(\alpha x) = max|\alpha x_i| = max|\alpha||x_i| = |\alpha| \cdot max|x_i| = |\alpha| f(x)$
3. $f(x+y) = max|x_i + y_i| \leq max(|x_i| + |y_i|) \leq max|x_1| + max|y_i| = f(x) + f(y)$

**Norma Uno**

$$f(x) = ||x||_1 = \sum_{i=1}^{n} |x_i|$$

**In generale le norme di uno stesso vettore hanno valori differenti**

**Esempio** $x = \begin{bmatrix} 2 \\ 3 \end{bmatrix}$

$||x||_1 = |2| + |3| = 5$

$||x||_2 = \sqrt{2^2 + 3^2} = \sqrt{4+9} = \sqrt{13}$

$||x||_\infty = 3$

**Distanza** Data $f : R^n \to R$ **norma**, allora possiamo definire la **distanza su** $R^n$ come

$$dist(x, y) = ||x - y|| = f(x - y)$$

1. $f(x) \geq 0$ e $f(y) \geq 0 \Rightarrow dist \geq 0$ e $(dist = 0 \Leftrightarrow x = y)$
2. $f(\alpha x) = |\alpha| f(x)$
3. $f(x + y) \leq f(x) + f(y)$

**Anche la distanza assume valori differenti su norme differenti**

**Esempio** $x = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$, $y = \begin{bmatrix} 2 \\ 3 \end{bmatrix}$

$dist(x, y) = ||x - y||_1 = ||\begin{bmatrix} -1 \\ -3 \end{bmatrix}||_1 = 4$

$dist(x, y) = ||x - y||_2 = ||\begin{bmatrix} -1 \\ -3 \end{bmatrix}||_2 = \sqrt{10}$

$dist(x, y) = ||x - y||_\infty = ||\begin{bmatrix} -1 \\ -3 \end{bmatrix}||_\infty = 3$

**Equivalenza topologica** Può succedere che $x$ e $y$ siano vicini in norma uno e lontani in norma 2? **No** per l'**equivalenza topologica delle norme**: quantitativamente differenti, ma qualitativamente danno gli stessi risultati.

$||x - y||_1$ è piccolo $\Rightarrow ||x - y||_2$ è piccolo

$||x - y||_\infty$ è grande $\Rightarrow ||x - y||_1$ è grande

### 2.1.2 Norma Matriciale

**Definizione** $f : R^{n\times n} \to R$ si dice **norma matriciale** se soddisfa le seguenti proprietà

1. $f(A) \geq 0 \wedge (f(A) = 0 \Leftrightarrow A = 0 \Leftrightarrow a_{ij} = 0 \ \forall i, j$
2. $f(\alpha A) = |\alpha| f(A)$
3. $f(A + B) \leq f(A) + f(B)$
4. $f(A \cdot B) \leq f(A) \cdot f(B)$

Difficile definire le norme metriciali, quind **costruiamo le norme matriciali a partire dalle norme vettoriali**.

**Definizione** Data $f_V : R^n \to R$ norma vettoriale, si dice **norma matriciale indotta** (**compatibile**) la norma $f_M : R^{n\times n} \to R$ definita da

$$f_M(A) = max_{\{x\in R^n \,|\, f_V(x)=1\}} \ f_V(Ax)$$

1. Prendo i vettori $x \in R^n \mid f_V(x) = 1$, in generale sono infiniti
2. Calcolare $Ax$, in generale infiniti vettori
3. Prendo $f_V(Ax)$, infiniti **valori**
4. Calcolo il **massimo** di questi valori

$||A||_1 = max_{||x||_1=1}||Ax||_1$

$||A||_2 = max_{||x||_2=1}||Ax||_2$

$||A||_\infty = max_{||x||_\infty=1}||Ax||_\infty$

**Esempio** $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \to ||A||_2 = max_{||x||_2=1}||Ax||_2$.
$||x||_2 = 1 \Leftrightarrow \sqrt{x_1^2 + x_2^2} = 1 \Leftrightarrow x_1^2 + x_2^2 = 1$ che è l'equazione per la circonferenza di centro 0 e raggio 1. Quindi tutti i vettori per cui $||x||_2 = 1$ sono tutti quelli la cui punta giace sulla circonferenza, quindi tanti quanti i punti della circonferenza **quindi infiniti**.

### 2.1.3 Teoremi di Caratterizzazione

La definizione quindi **non è pratica** (calcolabile) in generale, perché dovrei fare un massimo su infiniti termini. Quindi si introducono dei **teoremi di caratterizzazione**.

**Teorema 1** $||A||_2 = \sqrt{\rho(A^T A)}$
Dove $\rho(B) = max|\lambda_i|$ e $\lambda_i \in$ spettro$(B)$
$\rho(B)$ è il **raggio spettrale** di $B$, cioè il **modulo dell'autovalore di modulo massimo** (spettro$(B)$ è l'insieme degli autovalori).

**Teorema 2** $||A||_\infty = max_{i=1}^n \sum_{j=1}^n |a_{ij}|$ **somme per riga** poi **prendo il massimo**

**Teorema 2** $||A||_\infty = max_{j=1}^n \sum_{i=1}^n |a_{ij}|$ **somme per colonna** poi **prendo il massimo**

Da questi teoremi, se ne deriva che **calcolare la norma uno e la norma infinito è più semplice che calcolare la norma euclidea**.

**Vantaggi** delle norme matriciali → **ulteriorie proprietà** delle norme matriciali indotte

**Teorema** Sia $||\bullet||$ una norma vettoriale e $||-||$ la norma matriciale indotta.
Allora $\forall A \in R^{n\times n}$, $\forall v \in R^n$ si ha $||Av|| \leq ||A|| \cdot ||v||$

**Dimostrazione** Due casi

$v = 0$ $||Av|| = ||0|| = 0$
$||A|| \cdot ||v|| = ||A|| \cdot 0 = 0$

$v \neq 0$ $||Av|| = ||\ A\frac{v}{||v||} \cdot ||v||\ ||$ e chiamo il vettore $A\frac{v}{||v||} = z$. Quindi, per la seconda proprietà avrò
$= ||z|| \cdot ||v|| = ||A\frac{v}{||v||}|| \cdot ||v|| \leq ||A|| \cdot ||v||$

### 2.1.4 Condizionamento della risoluzione di un sistema lineare

$$Ax = b \longrightarrow \widehat{A}\widehat{x} = \widehat{b}$$

Per semplicità di analisi, $\widehat{A} = A$ e considero la perturbazione solo su $b$.

$$Ax = b \longrightarrow A\widehat{x} = \widehat{b}$$

Supponiamo anche A invertibile.
Misurare il condizionamento significa misurare quanto $x$ e $\widehat{x}$ sono vicini, attraverso

$$\epsilon_{in} = \frac{||x - \widehat{x}||}{||x||}$$

Studiamo la quantità

$$||x - \widehat{x}|| = ||A^{-1}b - A^{-1}\widehat{b}|| =$$

$$\widehat{b} = b + f \qquad b = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \rightarrow \widehat{b} = \begin{bmatrix} \widehat{b_1} \\ \widehat{b_2} \end{bmatrix} = \begin{bmatrix} b_1(1+\epsilon_1) \\ b_2(1+\epsilon_2) \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} + \begin{bmatrix} b_1\epsilon_1 \\ b_2\epsilon_2 \end{bmatrix} = b + f$$

$$= ||A^{-1}b - A^{-1}(b+f)|| = ||-A^{-1}f|| = ||A^{-1}f|| \leq ||A^{-1}|| \cdot ||f||$$

Quindi $||x - \widehat{x}|| \leq ||A^{-1}|| \cdot ||f||$
Siccome $||x||$ sta al denominatore, maggiorazione a denominatore usa una quantità più piccola.

$$Ax = b \Rightarrow ||Ax|| = ||b|| \Leftrightarrow ||b|| = ||Ax|| \leq ||A|| \cdot ||x||$$

Quindi $||x|| \geq \frac{||b||}{||A||}$
Da questo ottengo

$$\frac{||x - \widehat{x}||}{||x||} \leq \frac{||A^{-1}|| \cdot ||f||}{\frac{||b||}{||A||}} = ||A|| \cdot ||A^{-1}|| \cdot \frac{||f||}{||b||}$$

Quindi

$$\epsilon_{in} = \frac{||x - \widehat{x}||}{||x||} \leq ||A|| \cdot ||A^{-1}|| \cdot \frac{||\widehat{b} - b||}{||b||}$$

$\frac{||\widehat{b}-b||}{||b||}$ **perturbazione relativa sul termine noto**
$||A|| \cdot ||A^{-1}||$ **coefficiente di amplificazione**

Quindi se $||A|| \cdot ||A^{-1}||$ è grande, allora il sistem è mal condizionato.
Se $||A|| \cdot ||A^{-1}||$ è piccolo, allora è ben condizionato. Piccolo quanto?

**Osservazione** $||A \cdot A^{-1}|| = ||I|| \leq ||A|| \cdot ||A^{-1}||$, ma $||I|| = max_{||x||=1}||Ix|| = max_{||x||=1}||x|| = 1$
Quindi $||A|| \cdot ||A^{-1}|| \geq 1$ che quindi è un coefficiente di amplificazione effettivo, con valore minimo pari a 1.

## 2.2 Autovalori e Autovettori

**Autovalore** $\lambda \in C$ è **autovalore** di $A \in C^{n\times n}$ se $\exists x \neq 0 \mid A \cdot x = \lambda \cdot x$
Attenzione a $x \neq 0$.

**Autovettore** $x$ è detto **autovettore destro** relativo all'autovalore $\lambda$

Si osserva che

$$Ax = \lambda x \Leftrightarrow Ax - \lambda x = 0 \Leftrightarrow (A - \lambda I)x = 0$$

Quindi possiamo alternativamente dire

**Definizione** $\lambda \in C$ è **autovalore** di $A \in C^{n\times n} \Leftrightarrow det(A - \lambda I) = 0$

### 2.2.1 Polinomio caratteristico

$p(\lambda) = det(A - \lambda I)$ è **polinomio caratteristico** di A

**Definizione** $\lambda \in C$ è **autovalore** di $A \in C^{n\times n} \Leftrightarrow p(\lambda) = 0$ con $p(z) = det(A - zI)$ polinomio caratteristico di A
Quindi, per trovare gli autovalori un metodo è trovare il polinomio caratteristico e trovarne gli zeri.
Gli **autovalori sono gli zeri del polinomio caratteristico**.

**Osservazione** L'**autovettore non è univocamente determinato**, ma è **determinato a meno di una costante moltiplicativa**, quindi in generale sono infiniti autovettori per ogni autovalore.

$p(z) = det(A - zI)$ con $A \in C^{n\times n}$ è un **polinomio di grado** $n$. Possiamo esser anche più precisi: siccome è un polinomio il cui termine di grado più alto si ottiene moltiplicando gli elementi della diagonale principale, può essere scritto come

$$p(z) = (-1)^n z^n + p_{n-1} z^{n-1} + \ldots + p_1 z + p_0$$

**Teorema Fondamentale dell'Algebra** Un polinomio di grado $n$, su C, ha $n$ zeri eventualmente contati con la loro molteplicità algebrica. Quindi $p(z)$ su $C$ ha $n$ zeri $\lambda_1 \ldots \lambda_n$ eventualmente ripetuti.
Quindi possiamo scriverlo come

$$p(z) = (-1)^n \cdot \prod_{i=1}^{n} (z - \lambda_i)$$

che si può ulteriormente riscrivere mettendo insieme tutti gli autovalori uguali

$$p(z) = (-1)^n \cdot \prod_{i=1}^{k} (z - \lambda_i)^{\sigma_i}$$

con $k$ numero di zeri distinti, cioè di autovalori distinti $(\lambda_1 \neq \lambda_2 \neq \ldots \neq \lambda_k)$ e
$\sigma_i$ è detta **molteplicità algebrica dell'autovalore** $\lambda_i$.

**Molteplicità Algebrica** La **molteplicità algebrica** non è altro che il numero di **volte che l'autovalore si ripete** come zero del polinomio caratteristico.

**Molteplicità Geometrica** Se $\lambda$ è autovalore di $A$, allora $(A - \lambda I)$ è una **matrice singolare** $\Rightarrow$
$\tau = dim(Ker(A - \lambda I)) \geq 1$ e $\tau_i$ è la **molteplicità geometrica** dell'autovalore $\lambda_i$

**Teorema** La **molteplicità algebrica è maggiore o uguale della molteplicità geometrica**, cioè $\sigma_i \geq \tau_i$

### 2.2.2 Diagonalizzazione

**Definizione** $A \in C^{n\times n}$ si dice **diagonalizzabile** se $\exists\, S \in C^{n\times n}$ invertibile $\mid S^{-1} \cdot A \cdot S = D$ **diagonale**
Gli **autovalori di** $A$ **sono gli autovalori di** $D$, questo perché

$det(D-\lambda I) = det(S^{-1}AS-\lambda I) = det(S^{-1}AS-\lambda S^{-1}S) = det(S^{-1}(A-\lambda I)S) = det(S^{-1})\cdot det(A-\lambda I)\cdot det(S) = det(A-\lambda I)$

poiché $det(S) \cdot det(S^{-1}) = det(SS^{-1}) = det(I) = 1$. Quindi $D$ **e** $A$ **hanno lo stesso polinomio caratteristico, quindi stessi autovalori**.

**Teorema** $A \in C^{n\times n}$ **diagonalizzabile** $\Leftrightarrow \sigma_i = \tau_i$ per $i = 1 \dots k$, cioè **le molteplicità algebriche e geometriche sono coincidenti**.

**Esempi di matrici diagonalizzabili**

$k = n$, cioè la matrice ha tutti autovalori distinti. Quindi $\sigma_i = \tau_i = 1$ per $i = 1 \dots n$

(**Teorema Spettrale**) $A$ diagonalizzabile se $A = A^T$, cioè se $A$ è **simmetrica** $\Rightarrow$ **autovalori** $\in R$

**Diagonalizzare** $S^{-1}AS = D \Leftrightarrow AS = SD \Leftrightarrow AS \cdot e_j = SD \cdot e_j$ per $j = 1 \dots n$
Sappiamo che $S \cdot e_j$ corrisponde alla $j$-esima colonna di $S$, che chiamiamo $S_j$. Inoltre $D \cdot e_j$ è semplicemente $d_j \cdot e_j$ perché $D$ è diagonale e ha solo l'elemento $d_j$ sulla $j$-esima colonna.
Quindi $\Leftrightarrow AS_j = Sd_je_j = d_jSe_j = d_jS_j$
Questo ci dice che **le colonne di** $S$ **sono gli autovettori di** $A$.

**Calcolare gli autovalori di una matrice è, in generale, un problema difficile**, perché **gli autovalori non sono funzioni razionali dei coefficienti della matrice**.
Quindi **non esiste algoritmo che esegue un numero finito di operazioni razionali per calcolare gli autovalori**.
In generale si utilizzano **metodi iterativi** che **costruiscono successioni convergenti agli autovalori**. Per la convergenza di queste successioni, è **importante avere teoremi di localizzazione che mi dicono dove stanno gli autovalori nel piano complesso**.

### 2.2.3 Teorema di Gerschgorin

Sia $A \in C^{n\times n}$ e siano $K_i$ gli **insiemi** definiti da

$$K_i = \left\{ z \in C \;\middle|\; |z - a_{ii}| \leq \sum_{j=1, j\neq i}^{n} |a_{ij}| \right\} \quad per\ i = 1 \dots n$$

Allora se $\lambda \in C$ è autovalore di A

$$\Rightarrow \lambda \in \bigcup_{i=1}^{n} K_i$$

Gli autovalori di $A$ quindi appartengono ad almeno uno dei $K_i$, i cosiddetti **cerchi di Gerschgorin**, dove $a_{ii}$ è l'elemento della diagonale principale, $|z - a_{ii}|$ è il **centro** e $\sum_{j=1,j\neq i}^{n} |a_{ij}|$ è il **raggio**, formato dalla somma degli elementi della stessa riga escluso l'elemento della diagonale principale.

**Dimostrazione** $\lambda$ autovalore di $A$ se $\exists\, x \neq 0 \mid Ax = \lambda x$. Riscrivendo per componenti, otteniamo

$$Ax = \lambda x \Leftrightarrow \sum_{j=1}^{n} a_{ij}x_j = \lambda x_i \quad per\, i = 1 \dots n$$

Porto al primo membro

$$\Leftrightarrow \sum_{j=1, j\neq i}^{n} a_{ij}x_j = (\lambda - a_{ii})x_i \quad per\, i = 1 \dots n$$

$x \neq 0$, quindi esiste almeno una componente $\neq 0$. Prendiamo quella di modulo massimo $x_p$, quindi $|x_p| = ||x||_\infty$

$$Ax = \lambda x \Rightarrow \sum_{j=1, j\neq p}^{n} a_{pj}x_j = (\lambda - a_{pp})x_p \Rightarrow \left| \sum_{j=1, j\neq p}^{n} a_{pj}x_j \right| = |(\lambda - a_{pp})x_p|$$

Per le proprietà del valore assoluto

$$\Rightarrow |\lambda - a_{pp}| \cdot |x_p| \leq \sum_{j=1, j\neq p}^{n} |a_{pj}| \cdot |x_j|$$

$x_p \in C$ in generale, quindi posso dividere per $|x_p| \in R$, sicuramente $> 0$ e sicuramente positivo quindi la disequazione non cambia verso

$$\Rightarrow |\lambda - a_{pp}| \leq \sum_{j=1, j\neq p}^{n} |a_{pj}| \cdot \frac{|x_j|}{|x_p|}$$

Siccome $x_p$ è la componente di modulo massimo, allora tutte le $\frac{|x_j|}{|x_p|} \leq 1$ posso concludere che

$$\Rightarrow |\lambda - a_{pp}| \leq \sum_{j=1, j\neq p}^{n} |a_{pj}|$$

Per la definizione di $K_p$, questa relazione $\Rightarrow \lambda \in K_p$. Dato che non conosco $p$ in generale,

$$\Rightarrow \lambda \in \bigcup_{i=1}^{n} K_i$$

Abbiamo considerato i cerchi **per riga**. Ma posso anche considerare i cerchi per colonna, perché **gli autovalori di $A$ sono gli autovalori di $A^T$** in generale.
Questo perché $det(A - \lambda I) = det((A - \lambda I)^T) = det(A^T - \lambda I)$

**Esempio** Data $A = \begin{bmatrix} 3 & 1 & 0 \\ 1 & 3 & 1 \\ 0 & 1 & 3 \end{bmatrix}$
$K_1 = \{z \in C \mid |z - 3| \leq 1\}$
$K_2 = \{z \in C \mid |z - 3| \leq 2\}$
$K_3 = \{z \in C \mid |z - 3| \leq 2\} = K_1$

## 2.3 Fattorizzazione LU

**Problema** Dato $A \in R^{n\times n}, b \in R^n$, determinare $x \in R^n \mid Ax = b$. Assunzione: $A$ invertibile.
Quindi, convenzionalmente, $A$ è la **matrice dei coefficienti**, $b$ è il **termine noto** e $x$ il **vettore delle incognite**.
$x$ è **univocamente determinato** $x = A^{-1}b$ (rappresentazione teorica)
La **difficoltà è determinata dalla struttura di** $A$. Ci sono **alcuni casi più semplici**:

$A$ matrice **diagonale**: $A = a_{ij}$ con $a_{ij} = 0$ se $i \neq j$

Quindi $A = \begin{bmatrix} d_1 & 0 \ddots & 0 \\ 0 & \ddots & \ddots 0 \\ 0 & \ddots 0 & d_n \end{bmatrix}$ $A$ invertibile $\Leftrightarrow d_j \neq 0$ per $j = 1 \dots n$, questo perché $det(A) = \prod_{i=1}^{n} d_i$

$$\begin{bmatrix} d_1 & & \\ & \ddots & \\ & & d_n \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix} \Leftrightarrow \begin{cases} d_1 x_1 = b_1 \\ \vdots \\ d_n x_n = b_n \end{cases} \Leftrightarrow \begin{cases} x_1 = \frac{b_1}{d_1} \\ \vdots \\ x_n = \frac{b_n}{d_n} \end{cases}$$

Calcolabile in O(n) operazioni

```
for k = 1:n
        x(k) = b(k)/a(k, k);
end
```

$A$ **triangolare** superiore $= \begin{bmatrix} n & n & n \\ 0 & n & n \\ 0 & 0 & n \end{bmatrix}$ o **triangolare** inferiore $= \begin{bmatrix} n & 0 & 0 \\ n & n & 0 \\ n & n & n \end{bmatrix}$

**Triangolare superiore** se $a_{ij} = 0$ per $i > j$
**Triangolare inferiore** se $a_{ij} = 0$ per $i < j$
$A$ invertibile $\Leftrightarrow$ gli elementi sulla diagonale sono $\neq 0$
$det(A) = \prod_{i=1}^{n} a_{ii}$, quindi $det(A) \neq 0 \Leftrightarrow a_{ii} \neq 0$ per $i = 1 \dots n$

**Backward Substitution** Usato per risolvere un sistema triangolare superiore

$$A = \begin{bmatrix} a_{11} & \dots & a_{1n} \\ & \ddots & \vdots \\ 0 & & a_{nn} \end{bmatrix} \text{ con } a_{ij} = 0 \text{ se } i > j \rightarrow Ax = b \Leftrightarrow \begin{cases} \sum_{j=1}^{n} a_{1j}x_j = b_1 \\ \sum_{j=2}^{n} a_{2j}x_j = b_2 \\ \vdots \\ a_{nn}x_n = b_n \end{cases}$$

$$\sum_{j=k}^{n} a_{kj}x_j = b_k$$

$x_{k+1} \dots x_n \longrightarrow x_k$, quindi $a_{kk}x_k = b_k - \sum_{j=k+1}^{n} a_{kj}x_j$, da cui ricavo

$$x_k = \frac{1}{a_{kk}}(b_k - \sum_{j=k+1}^{n} a_{kj}x_j)$$

Di conseguenza $k$ lo scorriamo "**all'indietro**": `k = n:-1:1`

```
x(n) = b(n)/a(n, n)
for k = n - 1:-1:1
        s = 0
        for j = k + 1:n
                s = s + a(k, j)*x(j)
        end
        x(k) = (b(k) - s)/a(k, k)
end
```

**Costo computazionale**

$$\sum_{k=1}^{n-1}(n-k) = (n-1) + (n-2) + \ldots + 1 = \frac{n(n-1)}{2} = \frac{n^2}{2} + O(n)$$

Quindi ha un costo di O($n^2$) operazioni

**Forward Substitution** Usato per risolvere un sistema triangolare inferiore

$$A = \begin{bmatrix} a_{11} & & 0 \\ \vdots & \ddots & \\ a_{n1} & \dots & a_{nn} \end{bmatrix} \text{ con } a_{ij} = 0 \text{ se } i < j \rightarrow Ax = b \Leftrightarrow \begin{cases} a_{11}x_1 = b_1 \\ a_{21}x_1 + a_{22}x_2 = b_2 \\ \vdots \\ \sum_{j=1}^{n} a_{nj}x_j = b_n \end{cases}$$

**Matrice generica** Per una matrice generica $A \in R^{n \times n}$, l'**idea** è di **cercare di ridurre** $A$ **in una forma più semplice**, **mantenendo l'equivalenza dei sistemi lineari**.

$$Ax = b \longrightarrow Fx = g$$

con **sistemi equivalenti** e $F$ **più semplice di** $A$: ad esempio $F$ triangolare.
Cominciamo domandandoci se $A$ può essere riscritta come **prodotto di matrici triangolari**.

### 2.3.1 Fattorizzazione LU

$A = L \cdot U$ con $L$, $U$ triangolari

$$Ax = b \Leftrightarrow L \cdot Ux = b \Leftrightarrow \begin{cases} Ly = b \\ Ux = y \end{cases}$$

**Definizione** $A \in R^{n \times n}$ si dice **fattorizzabile** $LU$ se $\exists L$ matrice triangolare inferiore con elementi = 1 sulla diagonale principale e $U$ triangolare superiore tale che $A = L \cdot U$

**Esempio** $A = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ l & 1 \end{bmatrix} \begin{bmatrix} u_1 & u_2 \\ 0 & u_3 \end{bmatrix} = \begin{cases} 0 = u_1 \\ 1 = u_3 \\ 1 = l \cdot u_1 \\ 0 = l \cdot u_3 + u_2 \end{cases}$ ... non ci sono soluzioni

$\Rightarrow$ A non ammette fattorizzazione LU

**Esempio** $A = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ l & 1 \end{bmatrix} \begin{bmatrix} u_1 & u_3 \\ 0 & u_2 \end{bmatrix} = \begin{cases} 0 = u_1 \\ 1 = u_3 \\ 0 = l \cdot u_1 \\ 0 = l \cdot u_3 + u_2 \end{cases} = \begin{cases} u_1 = 0 \\ u_3 = 1 \\ 0 = 0 \\ l + u_2 = 0 \\ u_2 = -l \end{cases} = \begin{bmatrix} 1 & 0 \\ l & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 0 & -l \end{bmatrix}$

$\Rightarrow$ **infinite** fattorizzazioni $LU$, perché $l \in R$

**Esempio** $A = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ l & 1 \end{bmatrix} \begin{bmatrix} u_1 & u_3 \\ 0 & u_2 \end{bmatrix} = \begin{cases} 1 = u_1 \\ 1 = u_3 \\ 1 = l \cdot u_1 \Rightarrow l = 1 \\ 1 = l \cdot u_3 + u_2 \Rightarrow u_2 = 0 \end{cases}$

$\Rightarrow$ fattorizzazione $LU$ **unica**

**Teorema** Esistenza ed unicità della fattorizzazione
Sia $A \in R^{n \times n}$ e siano $A_k = A(1:k, 1:k)$ con $k = 1 \dots n$ le **sottomatrici principali di testa**.
Se $A_1, \dots, A_{n-1}$ sono invertibili $\Rightarrow \exists!\, LU$ di $A$

**Dim** Per induzione sulla dimensione della matrice
$n = 1$, $A = [a] \Rightarrow a = 1 \cdot a$

$$* = \left[\begin{array}{c|c} A_{n-1} & z \\ \hline v^T & \alpha \end{array}\right] = \left[\begin{array}{c|c} L_{n-1} & 0 \\ \hline x^T & 1 \end{array}\right] \left[\begin{array}{c|c} U_{n-1} & y \\ \hline 0^T & \beta \end{array}\right] \Leftrightarrow \begin{cases} A_{n-1} = L_{n-1}U_{n-1} \\ z = L_{n-1}y \\ v^T = x^T U_{n-1} \\ \alpha = x^T y + \beta \end{cases} \Leftrightarrow \begin{cases} y = L_{n-1}^{-1} z \\ x^T = v^T U_{n-1}^{-1} \end{cases}$$

Con $A_{n-1}$ matrice di ordine $n-1$
Le sue sottomatrici principali di testa di ordine $1 \dots n-2$ sono quelle di A e quindi, per ipotesi, invertibili
$\Rightarrow$ per ipotesi $\exists!$ fattorizzazione LU di $A_{n-1}$
Per ipotesi, $A_{n-1}$ invertibile, quindi

$$0 \neq det(A_{n-1}) = det(L_{n-1}U_{n-1}) =$$

Per Binet

$$= det(L_{n-1}) det(U_{n-1}) = det(U_{n-1})$$

Fine

**Matrice Freccia** ($\searrow$) Anche detta **arrow matrix** o **arrowhead matrix** $A = \begin{bmatrix} 1 & & & x_1 \\ & \ddots & & \vdots \\ & & 1 & \vdots \\ x_1 & \dots & \dots & x_n \end{bmatrix}$

# Capitolo 3

# Esercitazioni

## 3.1 Esercitazione 2

Studiare il **condizionamento** del problema, la **stabilità** degli algoritmi e il **costo computazionale** (stimare il numero di operazioni) di

$$f(x) = x^n = e^{n\log(x)}$$

con $x > 0$

**Condizionamento** Dato che non è dipendente dal modo in cui lo si calcola, calcoliamo il condizionamento per $f(x) = x^n$ con la formula $\epsilon_{in} = \frac{f'(x)}{f(x)} \cdot x \cdot \epsilon_x$

$$\epsilon_{in} = \frac{n\cdot \cancel{x}^{n-1}\cdot \cancel{x}}{\cancel{x}^n}\epsilon_x = n\epsilon_x$$

quindi

$$|\epsilon_{in}| = |n \cdot \epsilon_x| \leq n \cdot u$$

Dato che $C_x = n$ possiamo dire che il problema è **ben condizionato**

**Stabilità** Dobbiamo considerare i due algoritmi, perché algoritmi differenti producono errore algoritmico diverso.

$$f(x) = x^n = (((x \cdot x) \cdot x) \cdot x) \ldots) \cdot x$$

Non possiamo calcolare $x \cdot x$ ma calcoleremo $x \otimes x = x^2(1+\epsilon_1)$, assumendo $x \in$ F(B, t, m, M) per il calcolo dell'errore algoritmico.
Successivamente calcoliamo $x \cdot x \cdot x$, cioè $(x \otimes x) \otimes x = x^3 \cdot (1+\epsilon_1) \cdot (1+\epsilon_2)$.
Quindi $x \cdot \ldots \cdot x$ $n$ volte diventerà $x^n \prod_{i=1}^{n-1}(1+\epsilon_i)$. Quindi

$$g(x) = x^n \prod_{i=1}^{n-1}(1+\epsilon_i) = x^n(1+\sum_{i=1}^{n-1}\epsilon_i)$$

$$\epsilon_{alg} = \frac{g(x)-f(x)}{f(x)} \doteq \frac{x^n(1+\sum_{i=1}^{n-1}\epsilon_i)-x^n}{x^n} = \frac{x^n\sum_{i=1}^{n-1}\epsilon_i}{x^n} = \sum_{i=1}^{n-1}\epsilon_i$$

Quindi

$$|\epsilon_{alg} = |\sum_{i=1}^{n-1}\epsilon_i| \leq \sum_{i=1}^{n-1}|\epsilon_i| \leq (n-1)u$$

Quindi l'algoritmo è **numericamente stabile**. In questo caso, $\epsilon_{in}$ è dello stesso ordine di $\epsilon_{alg}$

<grafo dell'errore>

L'errore accumulato è $\delta_k = \epsilon_k + \delta_{k-1}$, si somma all'errore locale della moltiplicazione ($\epsilon_k$).

$$|\delta_k| = |\epsilon_k + \delta_{k-1}| \leq |\epsilon_k| + |\delta_{k-1}| \leq |\delta_{k-1}| + u \leq |\delta_{k-2}| + 2u \leq |\delta_1| + (k-1)u \leq |\delta_0| + ku = ku$$

Quindi possiamo dire $|\delta_n| \leq n \cdot u$. Notiamo che gli errori non si amplificano, perché è un operazione di moltiplicazione, ma vengono semplicemente sommati. La relazione trovata lo dimostra.

Per quanto riguarda il secondo algoritmo

$$f(x) = e^{n \cdot \log(x)}$$

Si osserva che **la funzione è razionale** ($f(x) = x^n$) ma **viene calcolata in macchina usando le funzioni non razionali** $h(x) = e^x$ e $g(x) = \log(x)$. In generale, quindi, non posso eseguire l'analisi dell'errore algoritmico con gli strumenti precedentemente usati. Questo **a meno che non si sappia com'è calcolata la funzione non razionale**. Aggiriamo il problema dicendo che

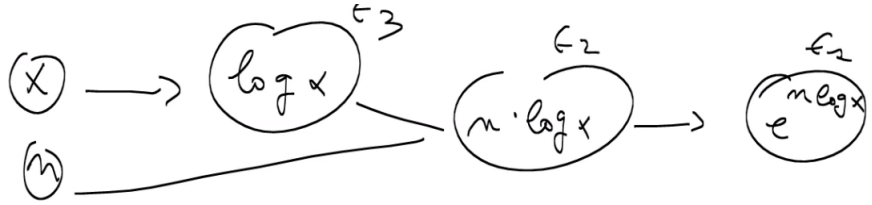
$$Exp(x) = e^x \cdot (1 + \epsilon_1)$$

con $|\epsilon_1| \leq u$ e

$$Log(x) = \log(x) \cdot (1 + \epsilon_2)$$

con $|\epsilon_2| \leq u$. Queste sono approssimazioni, in generale $|\epsilon_1|,\ |\epsilon_2| \leq k \cdot u$ con $k$ piccola che adesso trascuriamo perché l'analisi qualitativa è analoga.
In questo caso, trattiamo le funzioni non razionali come razionali.

**Analisi del grafo**




$|\epsilon_1| \leq u$, $|\epsilon_2| \leq u$, $|\epsilon_3| \leq u$
Per $x \to \log(x)$ allora $C_x = \frac{\frac{1}{x}}{\log(x)} = \frac{1}{\log(x)}$ ma non amplifica errore su $x$, perché per il calcolo dell'errore algoritmi assumiamo numeri di macchina.
Per $x \to e^x$ allora $C_x = \frac{e^x}{e^x} x = x$. Quindi $\epsilon_2$ ha come coefficiente di amplificazione pari a $n \cdot \log(x)$.
Di conseguenza calcolo l'errore algoritmico partendo dalla cima dell'albero e verso le radici (destra verso sinistra)

$$\epsilon_{alg} \doteq \epsilon_1 + (n \cdot \log(x))(\epsilon_2 + \epsilon_3)$$

Concludiamo che $|\epsilon_{alg}| = |\epsilon_1 + n \log(x)(\epsilon_2 + \epsilon_3)| \leq |\epsilon_1| + |n \log(x)(\epsilon_2 + \epsilon_3)| \leq u + |n \log(x)| \cdot |\epsilon_2 + \epsilon_3| \leq u + n|\log(x)|(|\epsilon_2| + |\epsilon_3|) \leq u + 2un|\log(x)| = u \cdot (1 + 2 \cdot n \cdot |\log(x)|)$ cioè

$$|\epsilon_{alg}| \leq u \cdot (1 + 2 \cdot n \cdot |\log(x)|)$$

Concludiamo che l'algoritmo è **numericamente instabile per $x$ piccolo o $x$ è grande**, ma è stabile in un intorno di 1.

L'analisi della stabilità ci dice che il primo algoritmo è preferibile.

**Costo computazionale** Il **primo algoritmo** $\Rightarrow (n-1)$ operazioni moltiplicative, quindi **O(n)**.
Il **secondo algoritmo**, più complicato perché ci sono due operazioni non razionali oltre alla moltiplicazione.

**Esercizio** Supposto $n = 2^k$ e $f(x) = x^n = x^{2^k}$, determinare un algoritmo che calcola $f(x)$ con $k = \log_2(n)$ operazioni moltiplicative. Eseguire anche l'analisi dell'errore algoritmico per l'algoritmo trovato.

## 3.2 Esercitazione 3

**Esercizio 1** $A = I + \alpha ee^T$ con $e^T = [1 \dots]^T$ vettore colonna, quindi $n \times 1$. $e \cdot e^T$ è quindi una matrice $n \times n$.
Quindi $A = I + \alpha \begin{bmatrix} 1 & \dots & 1 \\ & \dots & \\ 1 & \dots & 1 \end{bmatrix} = I + \begin{bmatrix} \alpha & \dots & \alpha \\ & \dots & \\ \alpha & \dots & \alpha \end{bmatrix}$

1. Per quali valori di $\alpha$, $A$ è invertibile?

2. Mostrare che l'inversa è $B = I + \beta ee^T$

3. Analizzare il condizionamento di $A$

**1.** Matrice è singolare $\Leftrightarrow Ax = 0$ ha una soluzione non nulla $\Leftrightarrow \begin{cases} x_1 + \alpha \sum x_i &= 0 \\ \dots & \\ x_n + \alpha \sum x_i &= 0 \end{cases} \Rightarrow x_1 = x_2 = \dots = x_n = x$
$x + \alpha \sum_{i=1}^{n} x_i = x + n\alpha x = x(1 + n\alpha)$
Se $1 + n\alpha \neq 0$, $x = 0 \Rightarrow x_1 = \dots = x_n = 0$
Se $1 + n\alpha = 0$, $\alpha = -\frac{1}{n}$
$A$ singolare $\Leftrightarrow \alpha = \frac{1}{n}$

**2.** $I = (I + \alpha ee^T)(I + \beta ee^T) \Leftrightarrow I = I + \beta ee^T + \alpha ee^T + \alpha ee^T \beta ee^T$ con $\alpha$, $\beta$ scalari quindi commutano
$\not{I} = \not{I} + \beta ee^T + \alpha ee^T + \alpha\beta e(e^T e)e^T$ con $e^T e$ scalare $= n \Leftrightarrow 0 = (\beta + \alpha + \alpha\beta n)ee^T$ con $ee^T \neq 0$
$\Leftrightarrow \alpha + \beta + \alpha\beta n = 0$
$\Leftrightarrow \beta = \frac{-\alpha}{1+\alpha n}$ con $\alpha \neq -\frac{1}{n}$
Quindi l'inversa è $I + \beta ee^T \Leftrightarrow \beta = \frac{-\alpha}{1+\alpha n}$

**3.** $A = \begin{bmatrix} \alpha+1 & \alpha & \dots & \alpha \\ \alpha & \alpha+1 & \alpha \dots & \alpha \\ \dots & & \dots & \end{bmatrix} ||A||_\infty \dots$
$K_\infty(A) = (|\alpha + 1| + (n-1)|\alpha|)(|\beta + 1| + (n-1)|\beta|)$
Mal condizionato per $|\alpha| \to +\infty$

**Esercizio 2** $A = I + \theta e_1 e_n^T + \theta e_n e_1^T$

# Capitolo 4

# Matlab

11 bit esponente rappresentato in traslazione, 52 bit mantissa (53 cifre rappresentabili per il bit nascosto) $\Rightarrow u = B^{1-t} = 2^{-52}$

## 4.1 Esponenziale

$f(x) = e^x$ approssimato con Taylor

$$e^x \simeq 1 + x + \frac{x^2}{2} + \frac{x^3}{3!} + \ldots + \frac{x^n}{n!}$$

Implementiamo un algoritmo che dati $x$, $n$ in input restituisce $y = 1 + x + \frac{x^2}{2} + \frac{x^3}{3!} + \ldots + \frac{x^n}{n!}$ in output.
Potrebbe essere utile un programma del tipo

```
y = 1;
for k = 1 .. n
        y = y + x^(k)/factorial(k)
end
```

Dal punto di vista del costo computazionale, però, ci sono dei problemi. Al passo $k$ faccio $k$ prodotti per $x^k$ e $k$ prodotti per $k!$, quindi **circa** $2k$ **prodotti**.
Il costo totale è del tipo $C = 2 + 2 \cdot 2 + 2 \cdot 3 + \ldots + 2 \cdot n = 2(1 + 2 + 3 + \ldots + n)$.
Ricordiamo $1 + 2 + \ldots + n = \frac{n(n+1)}{2}$, quindi $C = \cancel{2} \cdot \frac{n(n+1)}{\cancel{2}} = \mathrm{O}(n^2)$ quindi costo quadratico.
Una variazione più efficiente

```
y = 1; p = 1; m = 1;
for k = 1 .. n
        p = p * x;      % accumula le potente
        m = m * k;      % accumula il fattoriale
        y = y + p/m     % accumula la somma
end
```

Al passo $k$ ho 4 operazioni aritmetiche (o 3 operazioni moltiplicative, perché alcune volte si afferma che $t_* >> t_+$ in termini di operazioni tra bit (**costo booleano**))
Il costo totale $C = 4n$ operazioni (o $3n$).
I due algoritmi possono produrre numeri molto grandi. Una versione che argina questo problema è la seguente.

```
y = 1; z = 1;
for k = 1 .. n
        z = z * x/k;    % controllare l'esplosione dei numeri
        y = y + z;
end
```

**Codice MatLab** (file: `myexp.m`)

```
function [y] = myexp(x,n)
    % y = 1 + x + (x^2)/2 + ... + (x^n)/(n!)
    y = 1; z = 1;
    for k = 1:n
        z = z * x/k;
        y = y + z;
    end
end
```