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 base dei numeri in maniera univoca

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 ω. 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 ϵ_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 ϵ 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 δ_i sono analoghi agli ϵ_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 ϵ_{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**.

1.2 Matlab

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