1. Teoria degli errori

Si vuole stimare l'influenza degli errori sulla soluzione, ovvero di quanto la soluzione calcolata $\hat{y}$ si discosti da quella reale y.

Errore assoluto

Calcola la bontà dell'approssimazione in termini di cifre decimali.

$$E_a = |y - \hat{y}| \tag{1}$$

Errore relativo

Calcola la bontà dell'approssimazione in termini di cifre significative. È spesso rappresentato in percentuale ed utilizzato per avere una rapida intuizione della grandezza dell'errore.

$$E_r = \left| \frac{y - \hat{y}}{y} \right| \tag{2}$$

> L'errore relativo è una migliore stima della bontà dell'approssimazione, tranne quando y è prossimo a zero. Solitamente è più importante, poiché tiene in considerazione la grandezza dei numeri rispetto all'errore.

Approssimazioni corrette

Mentre con le cifre decimali ci riferiamo a tutte le cifre dopo la virgola, con **cifre significative** indichiamo le cifre necessarie ad esprimere il risultato di una misura con la precisione con cui è stata fatta.

- Diciamo che $\hat{y}$ è una corretta approssimazione di y a p cifre decimali se $E_a < 10^{-p}$
  - $x = 624.428731, y = 624.420711, E_a = 0.8 \times 10^{-2} < 10^{-2} \Rightarrow$ prime 2 cifre decimali uguali
- Diciamo che $\hat{y}$ è una corretta approssimazione di y a p cifre significative se $E_r < 10^{-p+1}$
  - $x = 624.428731, y = 624.420711, E_r = 1.3 \times 10^{-5} < 10^{-4} \Rightarrow$ prime 5 cifre significative uguali

Stime errore relativo

Il numero di cifre significative dà una stima dell'errore relativo. Consideriamo due numeri `990` e `110` con 2 cifre significative. L'errore sarà di ± 5, quindi avremo che $E_r(110) \approx 5\%$ $E_r(990) \approx 0.5\%$, quindi il range in cui varia l'errore relativo è $.5\nabla\cdot 5\%$. Si può stilare una tabella con la corrispondenza cifre significative - errore relativo.

Metodi iterativi

Lavoreremo spesso con metodi iterativi, che permettono di determinare una successione di soluzioni approssimate $\{x_n\}_n$ che converge alla soluzione esatta del problema x^*. Per ciascuna iterazione possiamo definire un errore come segue:

$$e_n = |x_n - x^*| \tag{3}$$

Il metodo si dice convergente se si ha che:

$$\lim_{n\to\infty} |e_n| = 0 \quad \text{oppure} \quad \lim_{n\to\infty} |x_n| = x^* \tag{4}$$

Ordine di convergenza

Dopo aver definito la nozione di convergenza di un metodo iterativo (def. dipendente dalla norma $|\cdot|$ utilizzata), si passa a definire l'ordine di convergenza. Diciamo che il metodo ha ordine di convergenza p se:

$$\lim_{n\to\infty} \frac{|e_{n+1}|}{|e_n|^p} = \text{costante} \quad \text{oppure} \quad |e_{n+1}| \leq c|e_n|^p \tag{5}$$

La seconda vale definitivamente a partire da un n sufficientemente grande e per qualche costante c. L'ordine di convergenza esprime il numero di cifre decimali che il metodo guadagna, ad ogni iterazione, rispetto alla soluzione esatta.

Altro sui metodi iterativi

Il metodo potrebbe iterare all'infinito, per cui bisogna stabilire un **criterio di arresto**. Per ogni tipo di problema è possibile progettare diversi algoritmi, che verranno valutati in base a:

- Stabilità
- Efficienza - Complessità computazionale
- Occupazione dello spazio

Esempio: Metodo di Gauss

Nel caso di un sistema lineare, il numero caratteristico del problema è il numero di equazioni del sistema che, in un sistema con matrice dei coefficienti quadrata, è uguale al numero delle incognite n. Nel caso del metodo di Gauss, che esamineremo più avanti, la complessità computazionale è $\frac{4}{3}n^3$.

Esempio: Schema di Horner

Lo schema di Horner serve ad abbassare il costo computazionale del calcolo di un polinomio. Si vuole eseguire il calcolo di:

$$p(x) = ax^3 + bx^2 + cx + d \tag{6}$$

Sono necessarie 3 somme e 6 moltiplicazioni per un totale di 9 operazioni floating point (flops). In generale, per calcolare:

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + \dots \tag{7}$$

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0 \tag{7}$$

Occorrono n somme ed $\frac{n(n+1)}{2}$ moltiplicazioni, per un totale di

$$n + \frac{n(n+1)}{n} = \frac{n(n+3)}{n} \sim n^2 \text{ flopls} \tag{8}$$

Se invece si applica lo **schema di Horner**:

$$p(x) = a_0 + x(a_1 + x(a_2 + x(\dots a_{n-1} + x a_n)\dots)) \tag{9}$$

Otteniamo n somme ed n moltiplicazioni per un totale di $2n$ flops.

Sistemi di numerazione

I sistemi di rappresentazione numerica sono posizionali, ovvero ogni cifra occupa una posizione corrispondente ad una potenza della base del sistema adottato. Una fonte di errore è data dal passaggio da un sistema di numerazione all'altro. Scelta una base b, ogni reale $a \in \mathbb{R}$ può essere scritto come

$$a = \pm(a_m b^m + \cdots + a_0 b^0 + a_{-1} b^{-1} + \dots) \tag{10}$$

Dove $0 \leq a_i \leq b-1$, $a_i \in \mathbb{N}$. La rappresentazione è univoca a meno che il numero non necessiti di infinite cifre consecutive $a_{-k} = b - 1$ (in decimale ad esempio: 0.1299999...), allora una rappresentazione equivalente consiste nel sopprimere la successione aggiungendo un'unità all'ultima cifra rimasta (dall'esempio precedente: 0.13).

Teorema

Sia $b \in \mathbb{N}$ e $b \geq 2$ la base, sia $x \in \mathbb{R} \setminus \{0\}$ un numero reale, allora **esiste unico** $e \in \mathbb{Z}$ ed una successione $\{a_i\}$ di numeri naturali $a_i \in \mathbb{N}$ tale che

$$x = \pm\left(\sum_{i=1}^{\infty} a_i b^{-i}\right) \cdot b^e \tag{11}$$

Dove

- $0 \leq a_i \leq b-1$
- $a_1 \neq 0$
- $a_i \neq b-1$ definitivamente

> Piccolo hint: sembra somigliare alla notazione scientifica.

Rappresentazione numerica in un calcolatore

In un calcolatore lo spazio di memoria è finito, e la sommatoria precedente si può estendere fino ad un t numero finito.

Rappresentazione in virgola fissa

In sintesi N_1 cifre prima della virgola, N_2 cifre decimali, $N = N_1 + N_2$ cifre totali. Un bit s è solitamente conservato per il segno.

Rappresentazione in virgola mobile

La posizione della virgola non è fissa, ma è data dall'esponente. Un numero di cifre t è riservato alla mantissa m, un numero N_e è riservato all'esponente e. Un bit s è solitamente conservato per il segno. I bit della mantissa m determinano la precisione con cui viene rappresentato un numero, mentre i bit dell'esponente e determinano il massimo ed il minimo numero rappresentabili.

Insieme dei numeri macchina

L'insieme dei numeri macchina F è l'insieme dei numeri reali che sono rappresentabili attraverso una mantissa da t cifre e il cui esponente sta tra due interi L (lower) ed U (upper) definiti nel calcolatore.

$$F(t, b, L, U) = \{0\} \cup \left\{ x \in \mathbb{R} : x = \pm b^e \sum_{i=1}^{t} a_i b^{-i} \right\} \tag{12}$$

Lo 0 è rappresentato a parte poiché ha una rappresentazione particolare. L'insieme F è finito e numerabile ed ha la seguente cardinalità:

$$|F| = 1 + 2(b-1)b^{t-1}(U - L + 1) \tag{13}$$

La rappresentazione non è unica, ma si dice normalizzata se $a_1 \neq 0$ e la mantissa $m \geq b^{t-1}$ (in pratica le regole utilizzate nella notazione scientifica).

Memorizzazione esponente

Tenendo conto che il lower bound L dell'esponente è un numero negativo (minimo numero rappresentabile) allora per conservare un esponente e si memorizza $e^* = e - L$, e dato che $e \geq L$ allora $e^* \geq 0$.

Conversione decimale-binario

La **parte intera** si divide per due: se c'è il resto si mette 1, altrimenti si mette 0 e si assegnano potenze di due crescenti.

La **parte decimale** si moltiplica per 2 (divide per 1/2), se il prodotto è minore di 1, allora si ha 0, altrimenti si ha 1. Quando il numero diventa maggiore di 1 si sottrae 1 procedendo come prima.

```
0.2 | 2 => 0 x 2^-1
0.4 | 2 => 0 x 2^-2
0.8 | 2 => 1 x 2^-3
1.6 | 2 => 1 x 2^-4
... (periodico)
0.2 = .0011
```

In questo caso abbiamo una rappresentazione finita in decimale ed una rappresentazione approssimata in binario, e ciò da luogo ad un errore di arrotondamento (round-off).

Conversione decimale a base qualunque

Sia n un numero in base 10 che vogliamo convertire in base $b \in \mathbb{N}$. Sia $n = (a_j a_{j-1} \dots a_1 a_0)_b$ la conversione attesa. Dividendo n per b sia ha:

$$\frac{n}{b} = a_j \cdot b^{j-1} + \ldots + a_1 \cdot b^0 + \frac{a_0}{b} \implies n = bn_0 + a_0 \tag{14}$$

Dove $n_0 < n$. Quindi l'ultima cifra della rappresentazione a_0 non è che il resto intero di n/b. Per ottenere la penultima cifra si divide n_0/b ed il procedimento si arresta ad a_j tale che $n_j = 0$.

Scegliere la rappresentazione in floating point

Dato $x \in \mathbb{R}$ come scelgo $fl(x) \in F$? Si hanno i seguenti casi:

- **Underflow** (esponente troppo piccolo) $e < L$
  - Si emette un warning e si pone $fl(x) = 0$
- **Overflow** (esponente troppo grande) $e > U$
  - Segnale di errore e arresto del programma
- Se $L \leq x \leq U$, allora si procede:
  - Se $a_k = 0$ per $k \geq t + 1$ allora il numero x è in F e $x = fl(x)$
  - Altrimenti si ha:
    - **(Chopping)** Si esclude la parte destra della t-esima cifra
    - **(Rounding)** Si arrotonda in base alla t-esima cifra

Epsilon macchina

Tutti i floating point che hanno lo stesso esponente e hanno una spaziatura $b^{e-t+1)}$, questo implica che due numeri con lo stesso esponente e consecutivi p_1 e p_2 sono legati dalla relazione $p_2 = p_1 + b^{e-t+1}$. La spaziatura cambia prima e dopo le potenze esatte della base. L'**epsilon macchina** è un upper bound dell'errore relativo che si ha arrotondando o troncando la rappresentazione reale per ottenere la rappresentazione in floating point. Dato che parliamo di errore relativo, possiamo studiare l'epsilon macchina limitandoci ad $e = 0$ e ai numeri positivi.

Utilizzando il rounding, l'errore assoluto più grande ottenibile è $b^{e-t+1}/2$, basta porre un numero reale al centro dello spacing. Il denominatore nell'errore relativo è il numero che stiamo approssimando. Per trovare un upper bound dell'errore relativo, questo numero deve essere il più piccolo possibile. I peggiori errori relativi capitano arrotondando numeri del tipo $1 + a$ dove $a \in [0, b^{-t+1}/2]$, nello specifico il peggior caso è $a = b^{-t+1}/2$. Calcolando l'errore relativo, il denominatore è $1 + a \approx 1$, quindi abbiamo un errore relativo pari a $b^{-t+1}/2$. Questo upper bound è proprio l'epsilon macchina. Utilizzando il chopping, il ragionamento si ripete ma considerando $a \in [0, b^{-t+1}]$, quindi l'epsilon macchina è proprio pari a b^{-t+1}.

Aritmetica Standard IEEE

Bits	Numeri	Epsilon Macchina	Range
32	$(-1)^s \cdot 2^{e-127} \cdot (1+m)$	2^{-24}	$2^{-126} \leq x \leq 2^{128}$
64	$(-1)^s \cdot 2^{e-1023} \cdot (1+m)$	2^{-53}	$2^{-1022} \leq x \leq 2^{1023}$

Wobbling precision

I numeri in floating point non sono equispaziati, ma si addensano in prossimità del più piccolo numero rappresentabile. All'interno dell'intervallo $[b^e, b^{e+1}]$ i punti sono equispaziati e la loro distanza è b^{e-t}. Ogni volta che si diminuisce o aumenta l'esponente e, diminuisce o aumenta la spaziatura. Il fenomeno è detto **wobbling precision** ed ha un andamento oscillatorio

Condizionamento e stabilità

> **Def.** Consideriamo il problema: trovare x tale che $F(x) = d$, dove d è il dato (o i dati) da cui dipende la soluzione x ed F è la relazione funzionale che lega x e d. Diremo che tale problema è **ben posto**, se per un certo dato, la soluzione esiste, è unica e dipende con continuità dai dati.

L'unicità e l'esistenza della soluzione sono problemi analitici, mentre la dipendenza è un problema numerico. La dipendenza continua dei dati significa che piccole perturbazioni dei dati danno luogo a piccole variazioni della soluzione, dove "piccolo" può essere inteso in senso relativo o assoluto. Nascono due problemi, ci si chiede:

1. Alterando i dati del problema, di quanto si altera la soluzione?
2. Come si propagano gli errori?

Numero di condizionamento

Il primo problema è connesso con la dipendenza continua dai dati della soluzione e può essere stimato con il **numero di condizionamento** del problema, numero che non dipende dall'uso dell'aritmetica finita del calcolatore, ma dal tipo di problema.

Def. Sia δd la perturbazione applicata ai dati, che scatena una variazione δx nella soluzione, quindi $d + \delta d \to x + \delta x$. Diremo K **numero di condizionamento relativo** il valore ottenuto come segue:

$$K = \frac{\|\delta x\| / \| x \|}{\|\delta d\| / \|d\|} \tag{15}$$

Se $x = 0, d = 0$ si calcola K_{ass} il **numero di condizionamento assoluto**:

$$K_{ass} = \frac{\|\delta x\|}{\|\delta d\|} \tag{16}$$

Se K è grande allora il problema è **mal condizionato**. Se un problema è ben posto ma K è grande basta riformulare il problema.

Propagazione dell'errore

Il problema (2) dipende dalla stabilità dell'algoritmo. Ad ogni problema numerico si possono associare più algoritmi.

> Un algoritmo è stabile se la propagazione degli errori dovuti all'aritmetica di macchina è limitata.

Un algoritmo è più stabile di un altro se in esso l'influenza degli errori è minore.

Errori dovuti a operazioni aritmetiche

In generale, i risultati di operazioni aritmetiche tra numeri macchina non sono numeri macchina. Indichiamo con $a \oplus b$ il valore reale di un calcolo, dove con $\oplus$ indichiamo una tra le 4 operazioni. Quando tale risultato non appartiene ai numeri macchina F, indichiamo la sua approssimazione con $fl(a \oplus b)$. Chiamiamo **round-off error** la differenza

$$a \oplus b - fl(a \oplus b) \tag{17}$$

Tale errore si può minimizzare con degli accorgimenti.

> **Esempio con somma.** Quando un calcolatore somma due numeri, porta il secondo numero allo stesso ordine del primo (stesso esponente) e somma la mantissa (da m cifre) in un accumulatore da $2m$ cifre, dopodiché aggiusta l'esponente. Questo può portare alla non validità della proprietà commutativa, distributiva ed associativa della somma e della moltiplicazione.
>
> **Esempio numerico.** Supponiamo $m = 4$ e di voler eseguire la somma $\sum_{i=1}^{11} x_i$, dove $x_1 = 0.5055 \times 10^4$ e per $i = 2, \dots, 11$ abbiamo $x_i = 0.4000 \times 10^0$. Nella prima somma avremo $x_1 + x_2$, quindi portiamo tutto all'ordine di x_1: $(0.5055 + 0.00004) \times 10^4$, ma essendo che la mantissa ha 4 cifre, la somma che viene calcolata è $(0.5055 + 0.0000) \times 10^4$ che rimane x_1. Questo non avviene calcolando prima la somma tra i termini $x_2, \dots, x_1 1$ e dopodiché x_1, quindi non vale la proprietà commutativa. Per minimizzare l'errore conviene sommare i numeri in base alla loro precisione, dal più piccolo al più grande (in valore assoluto).

Se $fl(a \oplus b)$ è arrotondato correttamente (rounding, come nel sistema IEEE), allora a meno di underflow o overflow si ha che:

$$fl(a \oplus b) = (a \oplus b)(1 + \delta) \qquad \text{con } |\delta| < \epsilon_M \tag{18}$$

In realtà, per evitare errori dovuti ad underflow si aggiunge una piccolissima quantità (10^{-45}) che da luogo ad una "eccezione" che viene segnalata da un flag.

Errori nelle 4 operazioni

Consideriamo l'errore nella forma precedente, per l'errore assoluto si ha:

- **Somma algebrica**: $x_1(1+\epsilon_1) \pm x_2(1+\epsilon_2)$
- **Prodotto**: $x_1(1+\epsilon_1) \cdot x_2(1+\epsilon_2)$
- **Divisione**: $x_1(1+\epsilon_1)/x_2(1+\epsilon_2)$

Per l'errore relativo si ha:

Prodotto:

$$\frac{x_1x_2 - x_1(1+\epsilon_1)x_2(1+\epsilon_2)}{x_1x_2} = -\epsilon_1 - \epsilon_2 - \epsilon_1\epsilon_2 \approx -\epsilon_1 - \epsilon_2 \tag{19}$$

Divisione:

$$\frac{\frac{x_1}{x_2} - \frac{x_1(1+\epsilon_1)}{x_2(1+\epsilon_2)}}{\frac{x_1}{x_2}} = \frac{-\epsilon_1 + \epsilon_2}{1+\epsilon_2} \approx -\epsilon_1 + \epsilon_2 \tag{20}$$

Somma algebrica:

$$\frac{(x_1 \pm x_2) - [x_1(1+\epsilon_1) + x_2(1+\epsilon_2)]}{x_1 \pm x_2} = \frac{-x_1\epsilon_1 - x_2\epsilon_2}{x_1 \pm x_2} \tag{21}$$

Pertanto abbiamo che:

- La **sottrazione** va bene per l'errore assoluto, ma si ha un errore relativo grande se $x_1 \approx x_2$.
- Il **prodotto** va bene per l'errore relativo, l'errore assoluto dipende dall'ordine di grandezza dei fattori.
- La **divisione** va bene per l'errore relativo, l'errore assoluto è grande se $x_2 \approx 0$.