Indice

Capitolo 1

Analisi degli Errori

Questi appunti riassumono le lezioni di teoria del corso di Calcolo Numerico a.a 2010-2011, docente: Francesca Mazzia

Le dispense sono distribuite con licenza Creative Commons Attribuzione-Non commerciale-Non Opere Derivate 2.5 Italy License `http://creativecommons.org/licenses/by-nc-nd/2.5/it/`.

1.1 Calcolo Scientifico

Il calcolo scientifico rappresenta l'insieme degli strumenti, delle tecniche e delle teorie necessarie per risolvere con il computer problemi della scienza e della tecnica. Il suo scopo principale è sviluppare metodi efficienti per calcolare approssimazioni di quantità che sono difficili o impossibili da ottenere per mezzi analitici.

Il calcolo scientifico nasce negli anni '40 con la modellizzazione di problemi balistici e di armi nucleari nella Seconda Guerra Mondiale; si sviluppa negli anni '70 e '80 per risolvere problemi industriali, es. progettazione di aerei; si consolida negli anni '90 su problemi di Informatica, Fisica, Chimica, Ingegneria, Economia, ...

Una buona parte del calcolo scientifico richiede la risoluzione di problemi matematici al calcolatore che può essere eseguita in modo efficiente solo se:

- si conosce lo strumento computazionale da usare.
- si conosce il problema da risolvere
- si costruisce un algoritmo che risolve il problema con una accuratezza desiderata ed entro i limiti delle risorse (tempo, memoria, ...) disponibili.

L' ambiente computazionale utilizzato per risolvere problemi matematici al calcolatore è costituito da:

- hardware: PC o supercomputer (computer vettoriali, computer paralleli);

- sistemi operativi e linguaggi: UNIX, LINUX, WINDOWS, ... C, C++, Java, Fortran95, ...
- strumenti per il data management: costruzione di database contenenti tutte le informazioni rilevanti per un particolare progetto applicativo;
- strumenti per la visualizzazione: rappresentazione grafica finalizzata ad una veloce comprensione del calcolo.

Il linguaggio di programmazione per la comunità scientifica è stato per molti anni il Fortran (introdotto negli anni '50). Ora anche altri linguaggi, specialmente il C e il C++ e Java, sono usati per il Calcolo Scientifico. La disponibilità di un linguaggio che può essere usato su una molteplicità di macchine determina la portabilità del software da macchina a macchina con pochissimi cambiamenti nelle performance del codice.

Ambienti che sono molto utilizzati per risolvere problemi del calcolo scientifico sono i Problem Solving Environments (PSE). Un PSE è un ambiente di alto livello che contiene tutto il software necessario per risolvere una determinata classe di problemi, fra cui:

- metodi avanzati per la soluzione;
- selezione automatica o semiautomatica dei metodi di soluzione;
- modi per incorporare facilmente nuovi metodi di soluzione.

Gli utenti possono usarlo senza avere una conoscenza specialistica del computer. Alcuni dei PSE più usati per il calcolo scientifico sono: Matlab; Maple; Mathematica; Scilab; Octave; R; Python; Maxima.

Costruire un algoritmo efficiente per il calcolo scientifico può essere una sfida complessa e difficile. Lo scopo di questo corso è farvi conoscere metodi e strumenti di base da poter utilizzare per risolvere problemi più complessi che potreste incontrare in futuro. In particolare ci soffermeremo su alcuni problemi di particolare importanza quando si lavora con i numeri al calcolatore fra cui: la rappresentazione dei numeri di macchina, il condizionamento di un problema e la stabilità degli algoritmi.

1.2 Il processo di risoluzione numerica

Per risolvere un problema del mondo reale al calcolatore si eseguono i seguenti passi:

- Problema reale → Modello Matematico: questo passo spesso richiede una semplificazione del modello, per esempio si suppongono trascurabili alcune grandezze fisiche
- Modello Matematico → Metodo Numerico: questo passo spesso genera delle approssimazione nel metodo risolutivo (es. un procedimento infinito approssimato mediante un procedimento finito: errore di troncamento);
- Metodo Numerico → programma: in questo passo si introducono gli errori di arrotondamento: i dati numerici elaborati e i risultati delle operazioni eseguite vengono arrotondati (si opera con aritmetica finita, errore di round-off)

Il processo di risoluzione numerica si occupa dello sviluppo di metodi numerici che tengano conto della natura del problema, delle risorse hardware e software, dello sviluppo di un algoritmo o un set di istruzioni che descrivono come risolvere un problema con un calcolatore. Questo richiede l'esame dei diversi aspetti che riguardano la scrittura, la comprensione, la valutazione di algoritmi e la loro implementazione in un determinato ambiente di calcolo; la formulazione di problemi test; lo sviluppo della fase di testing e della misura dell'efficienza. L' efficienza corrisponde alla ottimizzazione della complessità di tempo e di spazio e richiede uno studio accurato dell'algoritmo e della fase di implementazione attraverso la ricerca di tecniche che consentono l'effettiva minimizzazione dei calcoli e una corretta pianificazione delle aree di memoria da utilizzare.

Alcuni criteri per un buon codice sono:

- affidabilità;
- robustezza;
- portabilità;
- leggibilità;
- buona documentazione;
- ampia fase di testing.

1.3 Sorgenti di errori

Nel processo di risoluzione numerica è necessario essere capaci di controllare le diverse sorgenti di errori in modo che non interferiscano con i risultati calcolati. Di solito gli errori si propagano dalla loro sorgente a quantità che sono calcolate dopo, alcune volte con un fattore di amplificazione considerevole. Fra le diverse sorgenti di errori abbiamo:

- **Errori nei dati di input:** sono errori sistematici che dipendono dallo strumento di misura utilizzato per rilevare i dati di input o errori casuali che dipendono dall'ambiente di sperimentazione.
- **Errori di arrotondamento:** π non è rappresentato esattamente nel nostro computer. Il prodotto esatto di due numeri con s cifre contiene $2s$ o $2s-1$ cifre e per essere rappresentato nel nostro calcolatore deve essere arrotondato. Gli errori di arrotondamento sono legati alla rappresentazione dei dati ed esecuzione delle operazioni in aritmetica finita.
- **Errori di troncamento:** questi errori si verificano quando un processo infinito viene troncato, per esempio una serie infinita viene troncata dopo un numero finito i termini.
- **Semplificazione del Modello:** passando dal problema reale al modello matematico, per esempio, si suppongono trascurabili alcune grandezze fisiche.

- **Errori Umani:** In tutti i lavori di tipo numerico bisogna aspettarci errori umani come: errori in calcoli manuali. Bisogna sempre considerare che anche i libri di testo possono contenere errori in tabelle e formule. Ci possono essere errori nei programmi, errori di battitura o scambio di operatori e spesso errori di macchina.

Bisogna essere abili nel verificare la correttezza dei risultati. Ci sono errori controllabili e incontrollabili, Gli errori nei dati di input e le semplificazioni del modello possono essere considerati incontrollabili. Gli errori di troncamento sono controllabili. Gli errori di arrotondamento sono controllabili fino ad un certo punto.

Se A è una quantità che vogliamo calcolare e $\tilde{A}$ è un'approssimazione di A, allora l'errore commesso è la differenza fra i due valori:

$$\text{errore} = A - \tilde{A};$$

L'errore assoluto è il valore assoluto dell'errore:

$$\text{errore assoluto} = |A - \tilde{A}|;$$

e l'errore relativo si ottiene normalizzando l'errore assoluto con il valore esatto, se $A \neq 0$:

$$\text{errore relativo} = \frac{|A - \tilde{A}|}{|A|}.$$

L'errore relativo è più significativo dell'errore assoluto. È ragionevole chiedere che l'errore relativo sia minore di un valore prefissato. Se conosciamo una maggiorazione dell'errore assoluto, cioè:

$$|A - \tilde{A}| < tol.$$

possiamo fare una stima del valore esatto:

$$\tilde{A} - tol \leq A \leq \tilde{A} + tol.$$

Se conosciamo una maggiorazione dell'errore relativo, cioè:

$$\frac{|A - \tilde{A}|}{|A|} < tol.$$

possiamo fare una stima del valore esatto:

$$\frac{\tilde{A}}{1 + tol} \leq A \leq \frac{\tilde{A}}{1 - tol}.$$

Se riteniamo accettabili approssimazioni in cui

$$\frac{|A - \tilde{A}|}{|A|} < 0.001,$$

allora siano $A = 123457$ e $\tilde{A} = 123500$, calcoliamo l'errore relativo:

$$\frac{43}{123457} = 0.00034,$$

A	$\tilde{A}$	Errore Assoluto	Errore Relativo
1	0.99	0.01	0.01
1	1.01	0.01	0.01
-1.5	-1.2	0.3	0.2
100	99.99	0.01	0.0001
100	99	1	0.01

Tabella 1.1: Esempi di errore assoluto e relativo.

e poichè l'errore è minore di 0.001, l'approssimazione è accettabile.

Siano invece $A = 341.5$ e $\tilde{A} = 300$, l'errore relativo è:

$$\frac{41.5}{341.5} = 0.121,$$

e quindi l'approssimazione non è accettabile.

Se due quantità sono approssimativamente uguali, useremo la notazione $\approx$ per indicare questa relazione.

Questa è una notazione ambigua. È vero che $0.99 \approx 1$? Forse si. È vero che $0.8 \approx 1$? Forse no. Sia h un parametro reale che tende a zero tale che $\lim_{h\to 0} A_h = A$ allora,

$$A_h \approx A$$

per ogni h "sufficientemente piccolo".

Sia n un parametro intero che tende all'infinito tale che $\lim_{n\to\infty} A_n = A$ allora,

$$A_n \approx A$$

per ogni n "sufficientemente grande".

Un modo per scrivere la derivata prima di una funzione è:

$$f'(x) = \lim_{h\to 0} \frac{f(x+h) - f(x)}{h}$$

Possiamo quindi concludere che per h sufficientemente piccolo

$$\frac{f(x+h) - f(x)}{h} \approx f'(x)$$

L'uguaglianza approssimata verifica le proprietà transitiva, simmetrica e riflessiva:

$$\begin{aligned} A \approx B, B \approx C &\rightarrow A \approx C \\ A \approx B &\rightarrow B \approx A \\ &\quad A \approx A \end{aligned}$$

Un'altra notazione è la notazione dell"'O grande", conosciuta come **ordine asintotico**. Supponiamo di avere un valore y e una famiglia di valori che lo approssimano y_h, Se esiste una costante $C > 0$, indipendente da h, tale che:

$$|y - y_h| \leq C|\beta(h)|,$$

per h sufficientemente piccolo, allora diciamo che:

$$y = y_h + O(\beta(h)) \qquad \text{per } h \to 0,$$

cioè $y - y_h$ è dell'ordine di $\beta(h)$, $\beta(h)$ è una funzione del parametro h tale che $\lim_{h\to 0} \beta_h = 0$. Ci concentriamo sul modo in cui l'errore dipende dal parametro h e ignoriamo dettagli meno importanti come il valore di C.

L'utilizzo è analogo se abbiamo una successione x_n che approssima x per valori di n grandi:

$$|x - x_n| \leq C|\beta(n)|, \qquad x = x_n + O(\beta(n))$$

Teorema 1.3.1 Teorema di Taylor *Sia $f(x)$ una funzione avente $n+1$ derivate continue su $[a, b]$ per qualche $n \geq 0$, e siano x,$x_0 \in [a, b]$. Allora*

$$f(x) = p_n(x) + R_n(x)$$

con

$$p_n(x) = \sum_{k=0}^{n} \frac{(x - x_0)^k}{k!} f^{(k)}(x_0)$$

e

$$R_n(x) = \frac{1}{n!} \int_{x_0}^{x} (x - t)^n f^{(n+1)}(t)dt.$$

Inoltre esiste un punto ξ_x tra x e x_0 tale che:

$$R_n(x) = \frac{(x - x_0)^{n+1}}{(n + 1)!} f^{(n+1)}(\xi_x)$$

Ritorniamo all'approssimazione della derivata prima di una funzione e vediamo quale è l'errore che commettiamo se la approssimiamo con il rapporto incrementale. Sappiamo che:

$$f'(x_0) \approx \frac{f(x_0 + h) - f(x_0)}{h} = f'_h(x_0)$$

per h sufficiantemente piccolo. Vogliamo calcolare come $f'_h(x_0)$ si avvicina a $f'(x_0)$. Usiamo il Teorema di Taylor, con $n = 1$, $x = x_0 + h$ per esprimere $f(x_0 + h)$:

$$f(x_0 + h) = f(x_0) + hf'(x_0) + \frac{h^2}{2} f''(\xi_h)$$

quindi

$$\begin{aligned} f'_h(x_0) = & \frac{f(x_0 + h) - f(x_0)}{h} \\ = & \frac{f(x_0) + hf'(x_0) + \frac{h^2}{2} f''(\xi_h) - f(x_0)}{h} \\ = & f'(x_0) + \tfrac{h}{2} f''(\xi_h) = f'(x_0) + O(h). \end{aligned}$$

La quantità $\tau(h) = \frac{h}{2} f''(\xi)$ si chiama errore di troncamento o errore di discretizzazione e dipende da h. Possiamo dire che l'errore va a zero come $O(h)$

$$f'(x_0) \approx \frac{f(x_0 + h) - f(x_0)}{h} = f'_h(x_0)$$

Questa formula per approssimare la derivata si chiama metodo alle differenze in avanti.

Consideriamo adesso il rapporto:

$$\frac{f(x+h) - f(x-h)}{2h}$$

Utilizzando il Teorema di Taylor abbiamo:

$$f(x+h) - f(x-h) =$$

$$= f(x) + hf'(x) + \frac{h^2}{2} f''(x) + \frac{h^3}{6} f'''(x) + \frac{h^4}{24} f^{(4)}(\xi_1)$$

$$-f(x) + hf'(x) - \frac{h^2}{2} f''(x) + \frac{h^3}{6} f'''(x) - \frac{h^4}{24} f^{(4)}(\xi_2) =$$

$$= 2hf'(x) + \frac{h^3}{3} f'''(x) + \frac{h^4}{24} (f^{(4)}(\xi_1) - f^{(4)}(\xi_2))$$

In definitiva abbiamo

$$\frac{f(x+h) - f(x-h)}{2h} =$$

$$= f'(x) + \frac{h^2}{6} f'''(x) + \frac{h^3}{48} (f^{(4)}(\xi_1) - f^{(4)}(\xi_2)) =$$

$$= f'(x) + O(h^2)$$

Questa tecnica per approssimare la derivata prima si chiama metodo delle differenze centrali. In questo caso l'errore va a zero come $O(h^2)$

1.4 Rappresentazione dei numeri

L'utilizzo in modo corretto del calcolatore per fare calcoli di tipo scientifico, richiede la conoscenza di come sono rappresentati i numeri e degli errori che derivano da questa rappresentazione che sono chiamati errori di arrotondamento. L'uso dei numeri reali richiede una attenzione particolare, essendo questi infiniti, mentre il calcolatore ci da la possibilità di rappresentarne solo un numero finito.

La nostra notazione per rappresentare i numeri è una notazione posizionale a base 10. Ciò significa che se scriviamo 123 intendiamo esprimere il numero:

$$1 \cdot 10^2 + 2 \cdot 10^1 + 3 \cdot 10^0.$$

Cioè la cifra che occupa la posizione più a destra deve essere moltiplicata per 10^0 (= 1), quella che occupa la posizione centrale per 10, ecc. Questa convenzione è ormai universalmente adottata, al contrario di altre "convenzioni" che quotidianamente usiamo per la comunicazione di nostre idee ed esperienze come, ad esempio, le lingue, gli alfabeti, le direzioni di scrittura, la musica, ecc.

Proprio per la sua universale adozione, la notazione posizionale a base 10 può apparire a molti come l'unica possibile. Ciò non è vero. Anzi, dal punto di vista storico risulta che tale rappresentazione si è affermata solo da pochi secoli. In precedenza erano state usate tante altre rappresentazioni, alcune in base 10 ma non posizionali (ad esempio quella romana), altre posizionali ma non a base 10 (ad esempio la sumerica). Torniamo alla rappresentazione attuale.

Un qualunque numero lo possiamo esprimere nella forma:

$$\pm\alpha_p\alpha_{p-1}\ldots\alpha_0.\alpha_{-1}\alpha_{-2}\ldots\alpha_{-q}\ldots \tag{1.1}$$

sottointendendo:

$$\pm\alpha_p\cdot 10^p+\ldots+\alpha_0\cdot 10^0+\alpha_{-1}\cdot 10^{-1}+\ldots+\alpha_{-q}\cdot 10^{-q}\ldots$$

I coefficienti α_i sono uno dei seguenti simboli: 0, 1, 2, ..., 9. Nulla vieta di usare al posto di 10 un altro numero, diciamo β. In tale caso la stringa (intesa come successione ordinata di simboli) descritta dalla (1.1) assumerebbe un altro significato. Infatti, supposto che i simboli α siano compresi tra 0 ed $\beta-1$, essa sarebbe equivalente a:

$$\pm\alpha_p\cdot\beta^p+\ldots+\alpha_0\cdot\beta^0+\alpha_{-1}\cdot\beta^{-1}+\ldots+\alpha_{-q}\cdot\beta^{-q}\ldots$$

Si ottiene così la rappresentazione posizionale di un numero in base β.

Nel campo scientifico si preferisce usare una notazione teoricamente equivalente alla (1.1), cioè:

$$\pm d_0.d_1d_2\ldots d_t\ldots\cdot\beta^q \tag{1.2}$$

con $d_0\neq 0$. Quindi il numero $3.57\cdot 10^2$ è espresso in notazione esponenziale normalizzata, mentre i numeri $73.54\cdot 10^2$ o $0.57\cdot 10^{-3}$ non lo sono. Con questa rappresentazione si possono rappresentare tutti i numeri eccetto lo zero. Essa non è univoca. Ad esempio in base 10 i due numeri:

$$1.200.....$$
$$1.199.....$$

rappresentano lo stesso numero. La parte $d_0.d_1d_2\ldots d_t\ldots$ viene chiamata mantissa e noi la indicheremo con m. Il numero q viene chiamato esponente, mentre β è la base. Ogni numero reale non nullo sarà quindi indicato in questa rappresentazione, detta *rappresentazione normalizzata*, con:

$$\pm m\cdot\beta^q$$

Il numero zero sarà rappresentato con 0. Ovviamente nella rappresentazione di un numero intero tutti i valori d_i saranno nulli da un certo indice (maggiore di uno) in poi. È ovvio che:

$$1 \leq m < \beta. \tag{1.3}$$

Una stessa stringa rappresenta numeri diversi a seconda della base usata. Ad esempio la stringa 123 in base quattro rappresenta il numero:

$$1 \cdot 4^2 + 2 \cdot 4 + 3 = \text{ventisette}$$

cioè il numero che in base 10 indichiamo con 27. Nel caso in cui β fosse più grande di dieci, il numero di simboli che usualmente si usano per indicare le cifre tra 0 ed $\beta - 1$, cioè 0,1,2,...,9 non bastano più. Si fa ricorso alle lettere maiuscole dell'alfabeto per integrare i simboli mancanti. Ad esempio se $\beta = 16$, i simboli usati sono 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F. Intendendo con questo che il numero dieci viene rappresentato con A, l'undici con B, ecc.. Il numero sedici sarà ovviamente indicato con 10.

Esiste una convenienza nell'uso di una base al posto di un'altra? Diamo uno sguardo alla tabella seguente in cui il numero (non la stringa!) centoventitrè è rappresentato in diverse basi:

base	rappresentazione
16	7B
10	123
5	443
4	1323
2	1111011

È evidente che lo stesso numero richiede un numero di cifre crescente al diminuire della base. Ciò può risultare scomodo. Vi è però certamente un vantaggio nell'usare le basi più piccole. Infatti i calcoli risultano semplificati. La tavola pitagorica, che in base dieci è formata da un quadrato di 100 elementi, nella base 2 risulterebbe formata da 4 elementi.

La base due è ancora più interessante perchè in tale base tutti i numeri sono rappresentati da successioni dei due simboli 0 ed 1 che possono essere messe in corrispondenza con ogni sistema che abbia solo due stati (ad esempio acceso-spento, magnetizzato-smagnetizzato ecc.). Difatti è per questo motivo che essa risulta indispensabile per la codifica di informazioni nei calcolatori. Ogni informazione, per essere memorizzata nel calcolatore, viene opportunamente codificata nel linguaggio comprensibile alla macchina che è appunto basato sui simboli del sistema di numerazione binario, 0 ed 1. Abbiamo già avuto occasione di dire che questi due simboli rappresentano lo stato fisico dei circuiti di cui è costituita la memoria, che possono essere spenti (=0) o accesi (=1).

Ovviamente la rappresentazione (1.2) necessita un numero infinito di cifre e quindi sarebbe impossibile memorizzarla in un calcolatore. Nei calcolatori ad ogni numero viene riservato uno spazio fisso di memoria, per cui sia il numero di cifre della mantissa che

il numero di cifre dell'esponente non possono superare dei limiti prefigurati. I numeri di macchina, detti anche numeri *floating point*, sono del tipo:

$$\pm d_0.d_1d_2\dots d_t \cdot \beta^q \tag{1.4}$$

con $d_0 \neq 0$, $0 \leq d_i \leq \beta - 1$ e $q_{min} \leq q \leq q_{max}$, più, naturalmente, lo zero. Denotiamo con realmin il più piccolo numero positivo e con realmax il più grande numero positivo rappresentabile.

Se per esempio $t = 1, \beta = 10, q_{min} = -2, q_{max} = 2$ allora:

$+1.0 \cdot 10^{-2}$ è il più piccolo numero positivo rappresentabile in notazione esponenziale normalizzata. I numeri successivi sono $1.1 \cdot 10^{-2}, 1.2 \cdot 10^{-2}, 1.3 \cdot 10^{-2}, \dots 2.0 \cdot 10^{-2}, \dots$, Il più grande numero rappresentabile in questa notazione è $9.9 \cdot 10^2$. Quanti numeri possiamo rappresentare in totale? Ogni numero è individuato dal segno, dalle due cifre d_0, d_1 e dall'esponente e. Ci sono 9 possibilità per $d_0 \in 1, 2, 3, \dots, 9$, 10 possibiltà per $d_1 \in 0, 1, , 2, \dots, 9$ e 5 possibilità per $q \in -2, -1, 0, 1, 2$. Quindi in totale abbiamo 900 numeri più lo zero.

La quantità che devo aggiungere ad 1.0 per avere il successivo numero di macchina è $\epsilon_M = 0.1$, tale numero si chiama ϵ di macchina.

Un numero reale rappresentato tramite la (1.2) deve essere prima espresso in cifre binarie 0 ed 1 per poter essere memorizzato in un calcolatore.

1.5 Errore assoluto e relativo

Consideriamo il numero reale positivo:

$$x = d_0.d_1d_2\dots d_t \dots \cdot \beta^q$$

Se consideriamo i numeri di macchina con t cifre per la mantissa, x sarà compreso fra:

$$x_1 = d_0.d_1d_2\dots d_t \cdot \beta^q$$

e

$$x_2 = x_1 + 0.0\dots01 \cdot \beta^q = x_1 + \beta^{-t} \cdot \beta^q$$

graficamente si ha:

$$\begin{array}{ccccc} & & d_0.d_1d_2\dots d_t\dots\cdot\beta^q & & \\ | & & | & & | \\ \hline x_1 & & & & x_2 \end{array}$$

Vi sono due modi diversi per approssimare questo numero mediante i numeri floating point, in cui solo t cifre della mantissa sono rappresentabili:

1. il troncamento;
2. l'arrotondamento.

x	troncato t=2	arrotondato t=2
5.672	5.67	5.67
-5.672	-5.67	-5.67
5.677	5.67	5.68
-5.677	-5.67	-5.68

Tabella 1.2: Esempi di troncamento e arrotondamento

Nel primo caso il numero x viene sempre approssimato con x_1 trascurando tutte le cifre successive a t; nel secondo caso il numero viene approssimato con x_1 se $x < (x_1 + x_2)/2$, cioè se x si trova nella prima metà dell'intervallo $[x_1 x_2]$, altrimenti viene approssimato con x_2. Se $x = (x_1 + x_2)/2$ allora x viene approssimato con x_1 se il numero intero $d_0 \dots d_t$ è pari, altrimenti viene approssimato con x_2. Vedi la Tabella 1.2 con alcuni esempi di arrotondamento e troncamento.

Denotiamo con $fl(x)$ l'approssimazione di x. Se viene usato il troncamento, qualunque sia la posizione di x nell'intervallo, x viene approssimato con il numero di macchina alla sua sinistra e quindi l'errore assoluto, cioè la distanza in valore assoluto di $fl(x)$ da x, sarà sempre minore dell'ampiezza dell'intervallo $[x_1,\ x_2]$:

$$|fl(x) - x| < \beta^{-t} \cdot \beta^q$$

Nel secondo caso invece l'errore assoluto sarà minore della metà dell'ampiezza dell'intervallo:

$$|fl(x) - x| < \frac{1}{2}\beta^{-t} \cdot \beta^q$$

L'errore relativo di un numero $x \neq 0$, approssimato con $fl(x)$ è definito da:

$$\left| \frac{fl(x) - x}{x} \right|$$

Teorema 1.5.1 *Se $x \neq 0$ e usiamo t cifre per rappresentare la mantissa, allora:*

$$\left| \frac{fl(x) - x}{x} \right| \leq u \tag{1.5}$$

dove:

$$u = \begin{cases} \beta^{-t} & \text{in caso di troncamento} \\ \frac{1}{2}\beta^{-t} & \text{in caso di arrotondamento} \end{cases}$$

Il numero u è detto unità di arrotondamento.

Dimostrazione. Sia $x = d_0.d_1 d_2 \dots d_t \dots \cdot \beta^q$, allora $|x| \geq \beta^q$ e, nel caso di troncamento, si ha:

$$\frac{|fl(x) - x|}{|x|} \leq \frac{\beta^{q-t}}{\beta^q} \leq \beta^{-t},$$

da cui segue la tesi.

Si procede analogamente nel caso di arrotondamento e nel caso di numeri negativi.

Come conseguenza del Teorema 1.5.1 si ha che l'errore relativo, piuttosto che l'errore assoluto, è legato alle cifre esatte di un numero x. Se $|x| < realmin$, allora si ha un errore detto errore di "underflow", oppure $fl(x)$ si pone uguale a 0 e quindi:

$$\left|\frac{x - fl(x)}{x}\right| = 1,$$

cioè l'errore relativo è molto più grande di u. Se $|x| > realmax$ il numero non è rappresentabile e si genera un errore detto di "overflow".

Poniamo:

$$\epsilon = \frac{fl(x) - x}{x}$$

Sappiamo che $|\epsilon| \leq u$. Dalla precedente si ricava:

$$fl(x) = x(1 + \epsilon), \tag{1.6}$$

cioè per ogni x tale che $realmin \leq |x| \leq realmax$ esiste un numero ϵ in valore assoluto minore od uguale ad u tale che la sua approssimazione mediante un numero di macchina può essere espressa dalla (1.6). Se vale la (1.5) possiamo dire che il numero è rappresentato con t cifre esatte.

1.6 Standard IEEE

Descriviamo più in dettaglio come viene rappresentato un numero usando lo standard floating point IEEE, che costituisce un insieme di regole definito dall'istituto degli ingegneri elettrici e elettronici per la rappresentazione ed l'elaborazione dei numeri floating point nei computer. Lo standard IEEE specifica esattamente cosa sono i numeri floating point e come sono rappresentati nell'hardware e ha 4 scopi principali:

1. rendere l'aritmetica floating point il più accurata possibile;
2. produrre risultati sensati in situazioni eccezionali;
3. standardizzare le operazioni floating point fra i computer;
4. Dare al programmatore un controllo sulla manipolazione delle eccezioni;

Il punto (1) si ottiene in due modi. Lo standard specifica esattamente come un numero floating point dovrebbe essere rappresentato nell'hardware e richiede che le operazioni (addizione,radice,...) siano il più accurate possibili. Per il punto (2) lo standard produce Inf (Infinito) per indicare che un risultato è più grande del massimo numero floating point rappresentabile e NaN (Not a Number) per indicare che un risultato non è definito. Prima dello standard la maggior parte dei computer avrebbero bloccato un programma in tali circostanze. Il punto (3) ha alcune conseguenze:

(i) I numeri floating point possono essere trasferiti fra due computer che utilizzano lo standard IEEE in binario senza perdere precisione;

(ii) I dettagli dell'aritmetica floating point dovrebbero essere conosciuti dal programmatore;

(iii) Lo stesso programma su computer diversi dovrebbe produrre lo stesso risultato, anche se questo non è vero in pratica.

Ricordiamo che l'unità base di informazione immagazzinata da un computer è il bit, una variabile il cui valore è o 0 o 1. I bit sono organizzati in gruppi di 8 chiamati byte. L'unità più comune è la parola di 4 byte (32 bit). Per accuratezze più alte è raddoppiata a 8 byte o 64 bit. Vi sono 2 possibili valori per un bit, $2^8 = 256$ possibili valori per un byte e 2^{32} differenti parole di 4 byte.

I due tipi di numeri rappresentati sono interi (fixed point) e reali (floating point). Un numero reale ha tipo float in c e real in Fortran. Nella maggior parte dei compilatori c un float ha per default 8 byte invece di 4. Il formato IEEE sostituisce la base 10 con la base 2.

L'aritmetica con gli interi è molto semplice. Vi sono $2^{32} \approx 4 \cdot 10^9$ interi a 32 bit che coprono l'intervallo da $-2 \cdot 10^9$ a $2 \cdot 10^9$. Addizione, sottrazione e moltiplicazione sono fatte esattamente se la risposta e compresa nell'intervallo. La maggior parte dei computer danno risultati imprevedibili se il risultato è fuori dal range (overflow). Lo svantaggio dell'aritmetica con gli interi è che non possono essere rappresentate le frazioni e l'intervallo dei numeri è piccolo.

Quando una parola a 32 bit è interpretata come un numero floating point il primo bit è il bit del segno, $s = +$ o $s = -$. I successivi 8 bit formano l'esponente e i rimanenti 23 bit determinano la mantissa. Vi sono 2 possibili segni, 256 esponenti (che variano da 0 a 255) e $2^{23} \approx 8.4$ milioni di possibili mantisse. Per avere gli esponenti negativi si usa una convenzione: lo zero e nella posizione 127, dopo ci sono i numeri positivi e prima i numeri negativi. Quindi in memoria viene rappresentato $q^* = q + 127$. Inoltre il primo bit della mantissa, che rappresenta d_0, è sempre uguale a 1, quindi non vi è necessità di memorizzarlo esplicitamente. Nei bit assegnati alla mantissa viene memorizzato m^* e $m = 1.m^*$. Un numero floating point positivo ha quindi il valore $x = +(1.m^*)_2 2^{q^*-127}$ e la notazione $(1.m^*)_2$ indica che $1.m^*$ è interpretata in base 2.

Per esempio il numero $2.752 \cdot 10^3 = 2572$ può essere scritto:

$$\begin{aligned} 2752 &= 2^{11} + 2^9 + 2^7 + 2^6 = \\ &= 2^{11}(1 + 2^{-2} + 2^{-4} + 2^{-5}) = \\ &= 2^{11}(1 + (0.01)_2 + (0.0001)_2 + (0.00001)_2)) = \\ &= 2^{11}(1.01011)_2 \end{aligned}$$

allora la rappresentazione di questo numero avrebbe segno + esponente $q^* = q + 127 = 138 = (10001010)_2$ e $m^* = (010110\ldots0)_2$.

Il caso $q^* = 0$ (che corrisponde a 2^{-127}) e il caso $q^* = 255$ (che corrisponde a 2^{128} hanno una interpretazione differente e complessa che rende la IEE diversa dagli altri standard.

Se $q^* = 0$ il valore del numero memorizzato è $x = \pm(0.m^*)_2 2^{-126}$. Questo è chiamato underflow graduale (l'underflow è la situazione in cui il risultato di una operazione è diversa

da zero ma è più vicina a zero di qualsiasi numero floating point). I numeri corrispondenti vengono chiamati denormalizzati. L'underflow graduale ha la conseguenza che due numeri floating point sono uguali se e solo se sottraendone uno dall'altro si ha esattamente zero.

Se escludiamo i numeri denormali allora il più piccolo numero positivo floating point è $a = realmin = q^{-126}$, il successivo numero positivo b ha $q^* = 1$ e $m^* = 0.0\cdots01$, la distanza fra a e b è $2^2 3$ volte più piccola, della distanza fra a e zero. Senza l'underflow graduale vi è un gap fra zero ed il numero floating point più vicino.

L'altro caso è $q^* = 255$ che ha due sottocasi, Inf se $m^* = 0$ e NaN se $m^* \neq 0$. Sia il c che molti PSE stampano Inf o NaN quando si stampa una variabile floating point che contiene questo risultato. Il computer produce Inf se il risultato di una operazione è più grande del più grande numero rappresentabile.

Operazioni invalide che producono come risultato NaN sono: Inf/Inf, $0/0$, $Inf - Inf$. Operazioni con Inf hanno il significato usuale: $Inf + finito = Inf$, $Inf/Inf = NaN$, $Finito/Inf = 0$, $Inf - Inf = NaN$.

È chiaro che l'accuratezza delle operazioni Floating Point è determinata dalla grandezza degli errori di arrotondamento. Questo errore di arrotondamento è determinato dalla distanza di un numero dal numero floating point più vicino. Eccetto che per i numeri denormalizzati, i numeri floating point differiscono di un bit, l'ultimo bit di m^*. Cioè $2^{-23} \approx 10^{-7}$ in singola precisione. Questo è l'errore relativo e non l'errore assoluto.

L'aritmetica IEEE in doppia precisione usa 8 byte (64 bit). Vi è un bit del segno, 11 bit per l'esponente e 52 bit per la mantissa. Quindi in doppia precisione $\epsilon = 2^{-52} \approx 2.22 \cdot 10^{-16}$. Cioè l'aritmetica in doppia precisione ha circa 15 digit decimali di accuratezza, invece di 7. Vi sono 2^{11} possibili esponenti che variano da 1023 a -1022. Il più grande numero è circa $10^{307} \approx 2^{1023}$

Riassumendo:

IEEE - singola precisione. Occupa 4 byte = 32 bits

	$\longleftarrow$ 8 $\longrightarrow$	$\longleftarrow$ 23 $\longrightarrow$ —
s	q	m
segno	esponente	mantissa

Esso rappresenta $(-1)^s \cdot 2^{q-127} \cdot (1.m)$ Si noti che $d_0 = 1$ nella mantissa non deve essere esplicitamente memorizzato, quindi $m = d_1 d_2 \ldots d_t$. Range di numeri positivi normalizzati da 2^{-126} a $2^{127} \times 1.11\cdots1 \approx 2^{128}$ (da 1.2×10^{-38} a 3.4×10^{38})

IEEE - doppia precisione. Occupa 8 byte = 64 bits

	$\longleftarrow$ 11 $\longrightarrow$	$\longleftarrow$ 52 $\longrightarrow$ —
s	q	m
segno	esponente	mantissa

Esso rappresenta $(-1)^s \cdot 2^{q-1023} \cdot (1.m)$
Range di numeri positivi normalizzati da 2^{-1022} a $2^{1023} \times 1.11\cdots1 \approx 2^{1024}$ (da 2.2×10^{-308} a 1.8×10^{308})

IEEE - Eccezioni aritmetiche e risultati di default

tipo di eccezione	esempio	risultato di default
operazione invalide	$0/0 \quad 0 \times \infty$ $\sqrt{-1}$	NaN (Not a number)
Overflow		$\pm\infty$
Divisione per zero	Numero finito non nullo/0	$\pm\infty$
Underflow		numeri denormali

1.7 Operazioni con i numeri di macchina

Sia $o \in \{+, -, \times, /\}$ e siano x e y due numeri floating point. È improbabile che l'esatto valore di xoy sia un numero floating point.

Esempio $t = 3, \beta = 10$

$$1.111 \cdot 10^3 \times 1.111 \cdot 10^2 = 1.234321 \cdot 10^3$$

che richiede più di tre cifre decimali.

Il computer dovrebbe eseguire le operazioni aritmetiche di base in modo che il risultato finale sia il risultato esatto arrotondato al più vicino numero floating point. Il modello delle operazioni aritmetiche richiesto dall'IEEE è il seguente:

$$fl(xoy) = (xoy)(1 + \epsilon), \qquad |\epsilon| \leq u$$

o

$$fl(xoy) = (xoy)/(1 - \epsilon), \qquad |\epsilon| \leq u$$

L'aritmetica floating point IEEE richiede questo. Supponiamo di voler eseguire l'addizione tra due numeri $x = \alpha_0.\alpha_1\alpha_2 \ldots \alpha_t \cdot \beta^p$ e $y = d_0.d_1d_2 \ldots d_t \cdot \beta^q$, in cui $p > q$. Dobbiamo trasfomare i due numeri in modo da avere i due esponenti uguali a p. Per fare ciò bisogna che l'area di memoria in cui si eseguono le operazioni abbia uno spazio riservato alle mantisse superiore a quello usuale. Nel nostro caso si pone $y = (d_0.d_1d_2 \ldots d_t \cdot \beta^{q-p}) \cdot \beta^p = .0 \ldots 0d_0d_1 \ldots d_t \cdot \beta^p$ e poi si sommano le mantisse. Si procede quindi al troncamento o all'arrotondamento. In generale quindi il risultato di una operazione può considerarsi come il risultato esatto dell'operazione tra x ed y seguito dall'arrotondamento o dal troncamento.

Le operazioni con i numeri di macchina (aritmetica *floating point*), non godono di tutte le proprietà di cui godono le corrispondenti operazioni con i numeri reali. In generale non vale più la proprietà associativa. Consideriamo ad esempio il seguente caso in cui $\beta = 10$ e $t = 2$. Eseguiamo la somma $5.24 \cdot 10^{-2} + 4.04 \cdot 10^{-2} + 1.21 \cdot 10^{-1}$ in due modi diversi:

1) $5.24 \cdot 10^{-2} + (4.04 \cdot 10^{-2} + 1.21 \cdot 10^{-1}) = 5.24 \cdot 10^{-2} + (0.404 + 1.21)10^{-1} = 5.24 \cdot 10^{-2} + 1.61 \cdot 10^{-1} = (0.524 + 1.61)10^{-1} = 2.13 \cdot 10^{-1}$;

2) $(5.24 \cdot 10^{-2} + 4.04 \cdot 10^{-2}) + 1.21 \cdot 10^{-1} = 9.28 \cdot 10^{-2} + 1.21 \cdot 10^{-1} = (0.928 + 1.21)10^{-1} = 2.14 \cdot 10^{-1}$.

Supponiamo ora di avere due numeri reali x ed y e di volere fare la somma $x + y$. Nel calcolatore x sarà rappresentato da $fl(x) = x(1 + \epsilon_x)$ e y da $fl(y) = y(1 + \epsilon_y)$. Inoltre $fl(fl(x) + fl(y)) = (fl(x) + fl(y))(1 + \epsilon)$. Quindi $fl(fl(x) + fl(y)) = (x(1 + \epsilon_x) + y(1 + \epsilon_y))(1 + \epsilon)$ con $|\epsilon|, |\epsilon_x|, |\epsilon_y| \leq u$

Calcoliamo l'errore relativo trascurando i termini contenenti $\epsilon\epsilon_x$ ed $\epsilon\epsilon_y$,

$$\begin{aligned}
&\frac{|(x + y) - (fl(x) + fl(y))(1 + \epsilon)|}{|x + y|} \\
&= \frac{|x + y - (x + x\epsilon_x + y + y\epsilon_y)(1 + \epsilon)|}{|x + y|} \approx \\
&\approx |\epsilon| + |\epsilon_x|\frac{|x|}{|x + y|} + |\epsilon_y|\frac{|y|}{|x + y|}.
\end{aligned}$$

Se $x + y$ non è piccolo l'errore relativo è dello stesso ordine degli errori relativi $|\epsilon|$, $|\epsilon_x|$ ed $|\epsilon_y|$.Se $x + y$ è molto piccolo l'errore relativo è grande.

Dal precedente risultato risulta evidente che se il denominatore non è piccolo, come avviene certamente nel caso in cui i due numeri siano dello stesso segno, l'errore relativo è dello stesso ordine degli errori relativi ϵ, ϵ_1 ed ϵ_2 e quindi in valore assoluto minore di u. Ciò non avviene nel caso in cui il denominatore risulti molto piccolo. Ciò risulterà evidente nel seguente esempio.

Supponiamo infatti di voler effettuare la differenza tra i due numeri reali $x_1 = 0.147554326$ ed $x_2 = 0.147251742$ ed operare con $t = 5$ ed $\beta = 10$. Usando l'arrotondamento sarà $fl(x_1) = 1.47554 \cdot 10^{-1}$ e $fl(x_2) = 1.47252 \cdot 10^{-1}$. Si ha $fl(fl(x_1) - fl(x_2)) = 3.02000 \cdot 10^{-4}$, mentre la vera differenza è $3.02584 \cdot 10^{-4}$. Le ultime tre cifre della mantissa risultano errate. Ciò è dovuto al fatto che dopo aver eseguito la differenza $fl(x_1) - fl(x_2) = 0.000302$, la rappresentazione normalizzata $3.02000 \cdot 10^{-4}$ ha introdotto tre zeri alla fine della mantissa. Ovviamente questi tre zeri non sono esatti. L'errore relativo risultante è:

$$\frac{3.02584 - 3.02000}{3.02584} \approx 10^{-3}$$

che è piuttosto elevato. Quindi bisogna fare particolare attenzione a questo fenomeno, che viene chiamato *cancellazione numerica*. Essa si presenta quando si esegue la sottrazione tra numeri molto vicini.

Allo stesso modo si può analizzare il prodotto:

$$fl(fl(x) * fl(y)) = fl(x(1+e_x) * y(1+e_y)) = x(1+e_x)y(1+e_y)(1+e_*)$$

l'errore relativo è:

$$|x(1+e_x)y(1+e_y)(1+e_*) - xy|/|xy|$$

cioè

$$|(1+e_x)(1+e_y)(1+e_*) - 1|$$

semplificando ed eliminando i termini che contengono i prodotti di pi'**u**errori si ottiene:

$$\text{errore relativo nel prodotto } \approx |e_x| + |e_y| + |e_*|$$

Spesso è comodo vedere la (2.7) da un punto di vista leggermente diverso. Consideriamo per esempio il caso in cui *op* sia l'addizione. Sarà:

$$fl(x+y) = (x+\epsilon x) + (y+\epsilon y),$$

cioè il risultato dell'operazione di macchina può essere visto come l'operazione esatta su dati perturbati $(x+\delta x)$ e $(y+\delta y)$. Ovviamente nel caso considerato sarà $\delta x = \epsilon x$ e $\delta y = \epsilon y$. Nel caso in cui *op* sia la moltiplicazione si ha:

$$fl(xy) = xy(1+\epsilon) = (x+\delta x)(y+\delta y) \text{ con } \delta x = 0, \delta y = \epsilon y.$$

La tecnica di considerare la perturbazione nei dati che porterebbe allo stesso risultato finale se le operazioni fossero eseguite con precisione infinita è molto utile e viene chiamata *analisi degli errori all'indietro (backward)*.

Esercizio 1.7.1 Sia $t = 3$ e supponiamo che le operazioni siano eseguite con $\overline{t} = 6$ cifre decimali, eseguire le seguenti operazioni aritmetiche dopo over trasformato i numeri nella rappresentazione floating point:

$1.235 + 1.527 \cdot 10^{-4}$,

$.5 \cdot 10^3 - 2.5$,

$1.7435 \cdot 10^{-5} - 1.7448 \cdot 10^{-5}$.

Calcolare l'errore relativo e l'errore assoluto commesso nell'eseguire i calcoli.

1.8 Propagazione degli errori e condizionamento

Un problema numerico è una chiara e non ambigua descrizione della connessione funzionale fra i dati di input, variabili indipendenti del problema, e i dati di output, cioè i risultati desiderati. I dati di output devono essere determinati in modo univoco e devono dipendere in modo continuo dai dati di input. Un algoritmo è una descrizione completa di operazioni ben definite attraverso cui ogni insieme di dati di input ammissibile è trasformato nell'insieme dei dati di output. Le operazioni sono operazioni aritmetiche e logiche che un computer può eseguire.

La propagazione degli errori è di grande interesse per pianificare e analizzare esperimenti scientifici. Anche gli errori di arrotondamento di ogni passo di un calcolo si propagano sul risultato finale. Per molti algoritmi si può fare un'analisi degli errori di arrotondamento in cui si mostra che il risultato finale è il risultato esatto di un problema in cui i dati di input sono stati perturbati di poco.

Consideriamo, per esempio, una funzione $y = f(x)$ e supponiamo di volerla calcolare in un punto particolare $\overline{x}$ il quale è stato misurato con un errore Δx. Questo errore si ripercuote sulla variabile y con un errore Δy che, nel caso in cui f ammetta derivata prima in un intorno di $\overline{x}$ si può stimare facilmente. Si ha:

$$\overline{y} + \Delta y = f(\overline{x} + \Delta x) \simeq f(\overline{x}) + f'(\overline{x})\Delta x$$

da cui:

$$\frac{\Delta y}{\overline{y}} \simeq \overline{x}\frac{f'(\overline{x})}{f(\overline{x})}\frac{\Delta x}{\overline{x}}.$$

Si deduce che un errore relativo $|\Delta x|/|\overline{x}|$ sulla variabile indipendente produce un errore relativo sulla variabile dipendente che risulterà notevolmente amplificato se la quantità:

$$K(\overline{x}, f) = \left|\overline{x}\frac{f'(\overline{x})}{f(\overline{x})}\right|$$

è molto grande. Questa quantità è detta indice di condizionamento o numero di condizione del problema.

Definizione 1.8.1 *Dato un problema numerico $y = f(x) \in R^m$, $x \in R^n$, sia $\hat{x}$ fissato e assumiamo che $\hat{x}, f(\hat{x})$ siano diversi da zero. La sensibilità di y rispetto a piccoli cambiamenti di x si misura utilizzando il numero di condizione relativo:*

$$K(f, \hat{x}) = \lim_{\epsilon \to 0} \sup_{\|h\| \epsilon} \left\{ \frac{\|f(\hat{x} + h) - f(\hat{x})\|}{\|f(\hat{x}\|} / \frac{\|h\|}{\|\hat{x}\|} \right\}$$

dove $\|\cdot\|$ è una norma vettoriale,per esempio

$$\|x\|_p = (|x_1|^p + |x_2|^p + \ldots |x_n|^p)^(1/p), \qquad p = 1, 2$$

$$\|x\|_\infty = max_{1 \leq i \leq n}|x_i|$$

Il numero di condizione relativo misura la massima amplificazione di una perturbazione sui dati di input

$$\|\hat{y} - y\| \leq K\epsilon\|y\| + O(\epsilon^2)$$

quindi ci dobbiamo aspettar di avere circa $s = log_{10}(K)$ cifre significative in meno nella soluzione.

Se il numero di condizione è molto grande il problema si dice mal condizionato, se è di grandezza moderata si dice ben condizionato. Cosa significa molto grande? Dipende dall'accuratezza che richiediamo in output quando risolviamo un problema e dalla grandezza delle peturbazioni. Se $\epsilon = 10^{-16}$ e richiediamo 8 cifre significative corrette nella soluzione, allora K deve essere minore di 10^8 altrimenti il problema non è risolvibile nei limiti dell'accuratezza desiderata.

Se otteniamo una soluzione poco accurata per un problema mal condizionato non possiamo farci niente. Se fai una domanda sciocca, ottieni una risposta sciocca.

Esempio 1.8.1 Sia $f(x) = \log x$. Si ha $K(x, f) = 1/\log(x)$ da cui risulta che il calcolo del logaritmo è un problema mal condizionato se x è vicino ad 1.

Esempio 1.8.2 Consideriamo il problema di calcolare le radici di un polinomio. In questo caso i dati del problema sono i coefficienti del polinomio e il risultato finale le sue radici. Se consideriamo il seguente polinomio:

$$(x-1)^4 = x^4 - 4x^3 + 6x^2 + 4x + 1$$

sappiamo che la radice è 1 con molteplicità 4. Perturbiamo il termine noto, ottenendo il polinomio:

$$(x-1)^4 = x^4 - 4x^3 + 6x^2 + 4x + 1 - 10^{-8}$$

il quale ha le radici:

$$\begin{aligned} x_1 &= 1.01 \\ x_2 &= 0.99 \\ x_3 &= 1 + i0.01 \\ x_4 &= 1 - i0.01 \end{aligned}$$

quindi una pertubazione relativa di 1e-8 sul termine noto produce una perturbazione relativa di 1e-2 sul risultato. Possiamo concludere che il problema è mal condizionato.

Quando il risultato di un problema non si ottiene soltanto calcolando il valore di una funzione regolare in un punto, lo studio della propagazione degli errori e del condizionamento del problema può essere molto laboriosa.

Studiare il condizionamento del problema è molto importante nel caso in cui si voglia risolvere il problema usando il calcolatore. In questo caso infatti una perturbazione sui dati viene sempre provocata quando gli stessi vengono memorizzati.

1.9 Analisi della stabilità degli algoritmi

Di solito la poca accuratezza del risultato dipende dal mal condizionamento del problema, a volte, però, dipende dall'algoritmo. Ci sono algoritmi che eseguiti in aritmetica di macchina fanno propagare in maniera molto elevata gli errori di arrotondamento di ogni istruzione.

Definizione 1.9.1 *Un algoritmo si dice instabile se introduce grandi errori quando è applicato a un problema ben condizionato.*

Consideriamo un algoritmo con dati di input $x_1, x_2, \ldots, x_r$, $x = (x_1, x_2, \ldots, x_r)^T$ e dati di output $y_1, y_2, \ldots, y_n$, $y = (y_1, y_2, \ldots, y_n)^T$ e siano $\bar{y}_1, \bar{y}_2, \ldots, \bar{y}_n$, $\bar{y} = (\bar{y}_1, \bar{y}_2, \ldots, \bar{y}_n)^T$ l'output generato dall'algoritmo eseguito sul calcolatore. Possiamo fare due diversi tipi di analisi per studiare la stabilià di un algoritmo l'analisi forward (in avanti) e l'analisi backward (all'indietro).

- L'analisi Forward consiste nel trovare una maggiorazione dell'errore che si produce sull'output andando ad analizzare gli errori che si introducono nei singoli passi dell'algoritmo. Si determina, quindi una maggiorazione della quantità

$$\frac{\|\bar{y} - y\|}{\|y\|} \leq C_F u, \qquad \|y\| \neq 0$$

  con u l'unità di arrotondamento e C_F una costante non molto grande.

- L'analisi Backward consiste nel trovare $\bar{x}$ che, quando l'algoritmo viene eseguito in aritmetica esatta, genera $\bar{y}$ come risultato. Quindi si trova una maggiorazione dell'errore relativo:

$$\frac{\|\bar{x} - x\|}{\|x\|} \leq C_B u, \qquad \|x\| \neq 0$$

  con u l'unità di arrotondamento e C_B una costante non molto grande.

Come sono legate l'analisi forward e quella backward? Se K è il numero di condizione del problema si ha che

$$\frac{\|\bar{y} - y\|}{\|y\|} \leq K \frac{\|\bar{x} - x\|}{\|x\|} \leq K C_B u$$

ne segue che un algoritmo che è backward stabile è anche forward stabile. Ci sono però alcuni problemi per cui non si può fare l'analisi backward. Un esempio è il prodotto esterno fra vettori:

$$A = xy^T = \begin{pmatrix} x_1 y_1, & \ldots & x_1 y_n \\ x_2 y_1, & \ldots & x_2 y_n \\ \vdots & \ldots & \vdots \\ x_r y_1 & \ldots & x_r y_n \end{pmatrix}.$$

Quando eseguiamo il prodotto esterno al calcolatore abbiamo un errore per ogni elemento

$$fl(x_i y_j) = x_i y_j (1 + \delta_{ij}) \text{ con } |\delta_{ij}| < u$$

La matrice $\bar{A}$ ottenuta utilizzando l'aritmetica di macchina avrà elementi

$$\bar{A}_{ij} = x_i y_j + x_i y_j \delta_{ij}$$

dunque

$$\bar{A} = A + \Delta$$

con

$$\Delta_{ij} = x_i y_j \delta_{ij}$$

non è possibile trovare delle perturbazioni sui dati di input che generano $\bar{A}$. Quando l'insieme dei dati di input è più grande dell'insieme dei dati di output è molto probabile che l'analisi backward non si possa effettuare. In tal caso si esegue un'analisi mista.

Bibliografia

[1] F. Mazzia, D. Trigiante, Laboratorio di Programmazione e Calcolo, Pitagora Editrice, Bologna, 1992.

[2] P. Amodio, D. Trigiante, Elementi di Calcolo Numerico, Pitagora Editrice, Bologna, 1993.

[3] J. F. Epperson, Introduzione all'analisi numerica, teoria, metodi, algoritmi. McGraw-Hill, Milano.

[4] U.M. Ascher,Chen Greif, A First Course on Numerical Methods, 2006.

[5] G. Dahlquist, A. Bjork, Numerical Methods in Scientific Computing, Volume 1, SIAM 2008.