L'analisi degli errori

Dario A. Bini, Beatrice Meini,
Dipartimento di Matematica, Università di Pisa
a.a. 2016-2017

24 settembre 2020

Sommario

Questo modulo didattico contiene risultati relativi allo studio della propagazione degli errori e al loro controllo nello svolgimento di elaborazioni numeriche.

1 Introduzione

Nella risoluzione di problemi del mondo reale è frequente incontrare errori a vari livelli, molto spesso anche a nostra insaputa. Gli errori hanno varia natura e sono generalmente causati dalla "finitezza" delle risorse a nostra disposizione quali strumenti di misura e risorse di calcolo.

Ad esempio, le misure fatte con gli strumenti della tecnologia, quali misure di velocità, temperature, pressioni, non possono essere esatte. Infatti gli strumenti fisici forniscono approssimazioni del valore reale, molto accurate ma pur sempre approssimazioni. La finitezza delle risorse di calcolo è un'altra sorgente importante di errori.

Un esempio significativo a questo proposito è dato dalla rappresentazione dei numeri. Si pensi ad esempio di memorizzare il numero π in un computer mediante le cifre di una sua rappresentazione in qualche base. Ovviamente non possiamo memorizzare un numero infinito di cifre, visto che il numero di celle di memoria, seppur grande, è finito. Siamo quindi costretti a fare un troncamento di π introducendo conseguentemente un errore.

La stessa situazione si presenta anche in casi apparentemente innocui e per questo più insidiosi. Ad esempio, quando digitiamo in un qualche sistema di calcolo `x=0.1` intendendo la rappresentazione in base 10, quindi 1/10, il computer apparentemente memorizza nella sua memoria il valore 1/10 che viene associato alla variabile `x`. Se poi richiediamo al computer di mostrarci `x`, ci apparirà sullo schermo il valore `0.1`. Ad esempio, usando il linguaggio *Octave*[1] e scrivendo

```
x=0.1;
disp(x)
```

[1] Un manuale di Octave si trova sul Web in versione html e in versione pdf

si ottiene

```
0.10000
```

anche usando il formato a più cifre col comando `format long` si otterrebbe

```
0.100000000000000
```

Tutto sembra regolare e tranquillo, perché stupirsi?. Però, poiché la rappresentazione dei numeri fatta all'interno del nostro computer è in base 2 (ormai è così nella quasi totalità dei computer), e poichè il numero 1/10 in base 2 ha una rappresentazione periodica, il numero effettivamente memorizzato nella variabile `x` non è 1/10 bensì una sua approssimazione, molto precisa ma pur sempre un'approssimazione, ottenuta troncando lo sviluppo periodico della rappresentazione in base 2.

In altre situazioni è il problema stesso che vorremmo risolvere che non può essere risolto in modo esatto e quindi richiede una approssimazione. Si pensi ad esempio agli zeri di un polinomio di grado maggiore o uguale a 5. Sappiamo dalla teoria di Galois che non esiste alcuna espressione formale che ci permette di rappresentare questi zeri, nel caso generale, attraverso le sole operazioni aritmetiche ed estrazioni di radici. Se vogliamo avere informazioni sugli zeri dobbiamo necessariamente approssimarli.

In certi casi, come nella risoluzione di sistemi lineari, anche se la soluzione si può esprimere attraverso un numero finito di operazioni aritmetiche, è spesso più conveniente per ragioni di costo computazionale calcolarla in modo approssimato.

La presenza degli errori nella rappresentazione dei dati come pure gli errori sviluppati nello svolgimento dei calcoli a causa del troncamento dei risultati parziali può alterare in modo drammatico il risultato finale del calcolo. Nel sito Some disasters caused by numerical errors si può trovare una lista di catastrofi causata da un errato controllo della propagazione degli errori. Tra questi, l'esplosione dell'Ariane 5, il fallimento dei missili Patriot, l'affondamento della piattaforma Sleipner.

Diventa quindi di fondamentale importanza sviluppare una strumentazione adeguata di concetti e proprietà che ci permetta di controllare e dominare gli errori e la loro propagazione.

Nel seguito, se $\tilde{x}$ è una approssimazione di x denotiamo con $\tilde{x} - x$ l'*errore assoluto* e, se $x \neq 0$, denotiamo con $(\tilde{x} - x)/x$ l'*errore relativo*. Si osserva che nell'errore relativo si rapporta l'errore assoluto al valore del dato x per cui il suo valore va letto come una "percentuale" di errore. Ad esempio, un errore relativo tale che $|(\tilde{x} - x)/x| = 1$ significa un errore del 100%, cioè una approssimazione molto scadente.

Naturalmente siamo interessati agli errori che derivano da procedimenti matematici e quindi non è compito nostro trattare gli errori provenienti da misure fisiche. La sorgente di errore più importante per noi è quella causata dalla rappresentazione in base dei numeri fatta con un numero finito di cifre. Ci occupiamo di questo nei prossimi paragrafi.

2 Rappresentazione in base

Sia $B \geq 2$ un numero intero, vale il seguente risultato di rappresentazione:

Teorema 1 (di rappresentazione in base) *Per ogni numero reale $x \neq 0$ esistono unici un intero p ed una successione $\{d_i\}_{i\geq 1}$ con le seguenti proprietà*

1) $0 \leq d_i \leq B-1$,

2) $d_1 \neq 0$,

3) per ogni $k > 0$ *esiste un* $j \geq k$ *tale che* $d_j \neq B-1$

per cui

$$x = \text{segno}(x) B^p \sum_{i=1}^{\infty} d_i B^{-i} \tag{1}$$

La proprietà espressa dal precedente risultato assume una forma più familiare se scegliamo $B = 10$ e se conveniamo di allineare gli elementi della successione in una notazione posizionale come

$$x = \pm 0.d_1 d_2 d_3 \cdots \times 10^p$$

In questo modo il numero π può essere rappresentato da

$$\pi = 0.3141592 \cdots \times 10^1$$

dove il segno + è stato omesso.

L'intero B è detto *base della rappresentazione*. Gli interi d_i per $i = 1, 2, \ldots$, sono detti le *cifre della rappresentazione* mentre p è chiamato *esponente*. Il numero $\sum_{i=1}^{\infty} d_i B^{-i}$ viene chiamato *mantissa*.

La condizione 2 è detta di *normalizzazione* ed ha un duplice scopo. Da una parte serve a garantire l'unicità della rappresentazione visto che permette di evitare rappresentazioni equivalenti quali

$$0.3141592 \times 10^1, \quad 0.0003141592 \times 10^4, \quad 3141.592 \times 10^{-3}.$$

Dall'altra permette di memorizzare in modo più efficiente un numero reale. Ad esempio nella rappresentazione 0.0003141592×10^4 sono impiegate molte più cifre rispetto a 0.3141592×10^1. Infatti, se si usa la rappresentazione normalizzata, l'informazione contenuta nelle tre cifre nulle dopo il punto decimale è codificata in modo più compatto nell'esponente che occupa una sola cifra.

La condizione 3 stabilisce che non sono ammesse configurazioni in cui da un certo punto in poi tutte le cifre sono uguali a $B-1$. Ad esempio il numero $13/100$ può essere scritto come 0.13 oppure come $0.1299999 \cdots$. La seconda rappresentazione viene vietata per garantire l'unicità

3 Numeri floating point

Un computer, essendo una macchina finita, può memorizzare una quantità finita di cifre per cui non sono fisicamente rappresentabili configurazioni dotate di un numero infinito di cifre. Diventa allora necessario, nella definizione dei numeri utilizzabili in un computer, limitarsi a rappresentazioni dotate di un numero finito di cifre. Diamo allora la seguente

Definizione 1 Dati gli interi $B \geq 2$, $t \geq 1$, $m, M > 0$, l'insieme

$$\mathcal{F}(t, B, m, M) = \{0\} \cup \{\pm B^p \sum_{i=1}^{t} d_i B^{-i},\ \ d_1 \neq 0,\ 0 \leq d_i \leq B-1,\ -m \leq p \leq M\}$$

è detto insieme dei *numeri di macchina* o anche dei *numeri in virgola mobile* o dei *numeri floating point*.

Si osservi che lo zero non è rappresentabile nella forma $\pm B^p \sum_{i=1}^{t} d_i B^{-i}$ essendo $d_1 \neq 0$. Per cui viene inserito di ufficio nell'insieme $\mathcal{F}$ dei numeri di macchina.

Sia $x \neq 0$ un numero reale rappresentato come in (1) e si consideri

$$\tilde{x} = \text{segno}(x) B^p \sum_{i=1}^{t} d_i B^{-i}.$$

Se $-m \leq p \leq M$ il numero x viene ben rappresentato in $\mathcal{F}$ dal numero $\tilde{x}$, e in questo caso la quantità $\epsilon_x = (\tilde{x} - x)/x$, cioè *l'errore relativo di rappresentazione*, è tale che

$$\left| \frac{\tilde{x} - x}{x} \right| < B^{1-t}, \qquad \left| \frac{\tilde{x} - x}{\tilde{x}} \right| < B^{1-t}. \tag{2}$$

Infatti, per definizione di $\tilde{x}$ risulta

$$|\tilde{x} - x| = B^p \sum_{i=t+1}^{+\infty} d_i B^{-i} = B^{p-t-1} \sum_{i=0}^{+\infty} d_{t+1+i} B^{-i} < \frac{B^{p-t-1}(B-1)}{1 - B^{-1}} = B^{p-t},$$

dove la diseguaglianza stretta è ottenuta maggiorando tutte le cifre con $B-1$, configurazione che non può essere mai raggiunta per le ipotesi fatte. Inoltre, poiché $d_1 \neq 0$ risulta $|x| \geq B^p \times B^{-1}$, $|\tilde{x}| \geq B^p \times B^{-1}$. Ciò implica

$$|(\tilde{x} - x)/x| < B^{1-t}, \quad |(\tilde{x} - x)/\tilde{x}| < B^{1-t}.$$

Se invece $p < -m$ oppure $p > M$ allora il numero non è rappresentabile in $\mathcal{F}(t, B, m, M)$. Nel primo caso si dice che si è incontrata una situazione di *UNDERFLOW*. Nel secondo caso si dice che si è incontrata una situazione di *OVERFLOW*. Nel caso di underflow il numero ha un valore assoluto troppo

piccolo per essere rappresentato in $\mathcal{F}$. In certi sistemi numerici esso viene rappresentato da zero. Questo fatto è deprecabile poichè se $\tilde{x} = 0$, l'errore relativo di rappresentazione vale $|\tilde{x} - x|/|x| = 1$ cioè è un errore del 100%.

Si osservi che la (2) dice che, indipendentemente dal valore di $x \neq 0$, purché non si verifichino condizioni di overflow o di underflow, l'errore relativo di rappresentazione è limitato superiormente dalla costante B^{1-t}. Cioè il sistema di numeri floating point fornisce una limitazione *uniforme* all'errore relativo di rappresentazione.

Il numero B^{1-t} viene chiamato *precisione di macchina* e rappresenta il massimo livello di precisione di un sistema floating point. Nel seguito denoteremo con $u = B^{1-t}$ la precisione di macchina.

Il numero $\tilde{x}$ rappresenta x con la precisione data da t cifre in base B. Infatti si dice che $\tilde{x}$ ha t *cifre significative* poiché tutte le t cifre di $\tilde{x}$ concorrono nel dare la massima informazione su x. Questo fatto ci porta ad estendere il concetto di numero di cifre significative nel modo seguente. Se in generale y è una approssimazione di $x \in \mathbb{R}$ tale che $|(y-x)/x| < B^{1-c}$ si dice che y ha c *cifre significative* in base B. Ciò non implica che le prime c cifre della rappresentazione in base B di x e di y coincidono. Si considerino ad esempio con $B = 10$ e $c = 5$, i valori $x = 0.12000$ e $y = 0.11999$ in cui tutte e 5 le cifre sono significative ma non tutte coincidono. In ogni caso, se y approssima x con errore relativo $\epsilon = (y - x)/x$ possiamo dire che y ha $1 + \log_B |\epsilon|^{-1}$ cifre significative.

Si osservi ancora che due sistemi floating point in base B_1 e B_2 rispettivamente con t_1 e t_2 cifre hanno precisione di macchina rispettivamente $B_1^{1-t_1}$ e $B_2^{1-t_2}$. Per cui il primo è più preciso del secondo se $B_1^{1-t_1} < B_2^{1-t_2}$. Da cui $(1 - t_1) \log B_1 < (1 - t_2) \log B_2$, cioè

$$t_1 > (t_2 - 1)\frac{\log B_2}{\log B_1} + 1.$$

Ad esempio un sistema in base 2 con 53 cifre, come quello più diffuso sui computer in commercio, ha la stessa precisione di macchina di un sistema in base 4 con 27 cifre essendo $2^{1-53} = 4^{1-27}$. Inoltre, poiché $10^{-16} < 2^{1-53} < 2.2205 \times 10^{-16}$, la precisione di questo sistema fornisce almeno 16 cifre significative in base 10.

Il tipo di approssimazione di x con $\tilde{x}$ lo abbiamo ottenuto mediante *troncamento* della rappresentazione (1). Un'altra possibilità di rappresentazione consiste nel considerare l'*arrotondamento* di x, cioè il numero di macchina più vicino a x. In questo caso si può verificare che l'errore relativo di rappresentazione è minore o uguale a $\frac{1}{2}B^{1-t}$. Nel seguito trattiamo solo il caso in cui si considera il troncamento, il caso di arrotondamento può essere trattato in modo analogo.

3.1 Rappresentazione fisica

Generalmente le rappresentazioni floating point implementate sui computer sono fatte in base $B = 2$. In particolare così è lo standard IEEE che ritroviamo sui pc con processori INTEL, e AMD. Ad esempio, una rappresentazione in base 2 con 24 cifre viene realizzata nel modo seguente ed è chiamata precisione semplice.

- Il numero viene memorizzato su 32 bit (bit è la contrazione di binary digit, cifra binaria).
- Il primo bit più a sinistra memorizza il segno della mantissa. Se il bit è 0 allora la mantissa è intesa positiva; se il bit è 1 allora la mantissa è intesa negativa.
- Gli 8 bit successivi, procedendo da sinistra a destra, racchiudono l'informazione sull'esponente p. Più precisamente, se c è il numero intero corrispondente alla rappresentazione binaria degli 8 bit, allora l'esponente p viene inteso come $p = c - 127$. Ad esempio se gli 8 bit sono `10000011` che corrisponde al numero $1 + 2 + 2^7 = 129$, allora con questa convenzione l'esponente è $p = 2$.

  In questa rappresentazione lo standard IEEE fa due eccezioni: la configurazione di 8 bit nulli e quella di 8 bit uguali a 1 vengono usate per individuare situazioni di eccezione quali `NaN` (Not-a-Number), o `+Infty` e `-Infty`. Queste situazioni si incontrano quando viene chiesto al sistema di calcolo di eseguire operazioni vietate quali ad esempio $0/0$, $\sqrt{-1}$, o, rispettivamente $x/0$ con $x \neq 0$.
- I rimanenti 23 bit contengono le cifre $d_2, d_3, \ldots, d_{24}$. Si osservi che non importa memorizzare d_1 poiché, essendo $d_1 \neq 0$ e $d_1 < 2$ è necessariamente $d_1 = 1$.
- Lo zero viene rappresentato, rompendo la convenzione, mediante la configurazione di tutti bit nulli.

Si osservi che lo zero non sarebbe rappresentabile con la convenzione adottata che richiede $d_1 \neq 0$. Identificando lo zero con la configurazione di tutti bit nulli viene sacrificato il numero $B^{-1} \times B^{-m}$ che corrisponderebbe appunto a tale configurazione. Lo standard IEEE include anche altre codifiche di eccezione quali i *denormal numbers* che permettono di memorizzare numeri più piccoli in valore assoluto di 2^{-127}. Per valutare i limiti effettivi dell'esponente provate con Matlab o Octave a scrivere

```
a = 1023 ; 2^a
```

e ottenete

```
8.9885e+307
```

mentre ponendo `a = 1024;` ottenete `Inf`. Similmente, scegliendo `a = -1075;` si ottiene `0`, mentre con `a = -1074;` si ottiene `4.9407e-324`.

La precisione di macchina in Matlab e in Octave è indicata con `eps`. Infatti se digitiamo `log2(eps)` si ottiene il valore `-52`.

Lo standard IEEE prevede anche una rappresentazione su 64 bit chiamata precisione doppia e una rappresentazione su 128 bit chiamata precisione quadrupla. Le principali caratteristiche di queste rappresentazioni sono riportate nella tabella 1.

Nome	Bit esponente	Bit mantissa	Min/Max esponente	unit roundoff	Cifre base 10	Massimo esponente decimale
Precisione semplice	8	24	-127/128	1.2×10^{-7}	7.92	38.23
Precisione doppia	11	53	-1023/1024	2.2×10^{-16}	16.65	307.95
Precisione quadrupla	15	113	-16383/16384	1.9×10^{-34}	34.7	4931.77

Tabella 1: Rappresentazioni floating point in base 2: ripartizione dei bit tra esponente e mantissa e massimi valori ottenibili per l'esponente, la precisione, l'equivalente delle cifre decimali e l'equivalente dell'esponente in base 10. Nello standard IEEE effettivamente implementato alcune configurazioni di bit sono dedicate a codificare situazioni di eccezione quali `NaN`, `+Inf`, `-Inf` e altro, inoltre sono codificati i *denormal numbers* che permettono di memorizzare numeri di valore assoluto più piccolo del minimo altrimenti consentito. Per questo motivo i limiti dell'esponente sono leggermente diversi.

Una rappresentazione basata su 80 bit e adottata nei coprocessori matematici della serie Intel 8087 e successivamente nel Motorola 68881, è la rappresentazione estesa che usa 64 bit per la mantissa, 15 bit per l'esponente e un bit per il segno. Il numero di cifre della rappresentazione in questo caso è $t = 65$, e corrisponde a poco più di 19 cifre decimali. Il massimo e minimo esponente consentiti forniscono una copertura dei numeri positivi nel segmento $[3.65 \times 10^{-4951}, 1.19 \times 10^{4932}]$.

Sistemi di rappresentazione numerica a precisione variabile sono stati progettati per venire incontro a certe applicazioni, come quelle incontrate nei problemi di critto-analisi, in cui sono richieste alte precisioni di calcolo. Vale la pena citare uno dei pacchetti tra i più efficienti e di libero uso, coperto dalla licenza GNU, che è il GMP - GNU MultiPrecision Arithmetic Library, dove sono implementati gli algoritmi più sofisticati e veloci per la moltiplicazione di numeri dotati di molte cifre, quali il metodo di Karatsuba e l'algoritmo di Schoenhage-Strassen.

3.2 Aritmetica di macchina

Si osservi che se $a, b \in \mathcal{F}(t, B, m, M)$ non è detto che $c = a \,\mathrm{op}\, b$ appartenga ad $\mathcal{F}$, dove "op" è una delle quattro operazioni aritmetiche. Quindi per poter operare sui numeri di $\mathcal{F}$ dobbiamo introdurre una aritmetica approssimata nel seguente modo

$$\hat{c} = a\,[\mathrm{op}]\,b, \quad a\,[\mathrm{op}]\,b = \mathrm{tronc}(a \,\mathrm{op}\, b)$$

dove $\mathrm{tronc}(x)$ indica il troncamento del numero reale x a t cifre. In questo modo, se nello svolgere l'operazione aritmetica non si verificano situazioni di underflow

o di overflow, e $\hat{c} \neq 0$, allora per la (2) l'errore relativo $\delta = (\hat{c} - c)/c$ è tale che $|\delta| < u$. Analogamente per $\eta = (\hat{c} - c)/\hat{c}$ vale $|\eta| < u$ dove $u = B^{1-t}$ è la precisione di macchina. Possiamo quindi scrivere che

$$\hat{c} = (a \,[\mathrm{op}]\, b) = c(1+\delta) = c/(1+\eta), \qquad |\delta|, |\eta| < u. \tag{3}$$

L'errore relativo commesso δ (rispetto a c) o η (rispetto a $\hat{c}$) nel mantenere il risultato dell'operazione dentro l'insieme $\mathcal{F}$ viene chiamato *errore locale* generato dall'operazione floating point. Ciò definisce una aritmetica approssimata nell'insieme $\mathcal{F}$ che purtroppo non gode di molte delle proprietà formali algebriche. In particolare per l'aritmetica floating point non vale l'associatività delle operazioni e la distributività del prodotto rispetto alla somma. Inoltre ogni operazione aritmetica è potenzialmente sorgente di errori. Diventa quindi particolarmente importante *capire se e quando questi errori generati possono o meno alterare il risultato di un calcolo in modo significativo.*

4 Errori nel calcolo di una funzione

Supponiamo di avere assegnata una funzione $f : \Omega \subset \mathbb{R}^n \to \mathbb{R}$. Il nostro desiderio è quello di calcolare il valore di $f(x)$ per un valore assegnato di $x \in \Omega \subset \mathbb{R}^n$. Purtroppo dobbiamo accontentarci di calcolare $f(\tilde{x})$ dove $\tilde{x} \in \mathcal{F}^n \cap \Omega$ è una n-upla di numeri di macchina tali che $\tilde{x}_i = x_i(1+\epsilon_i)$, dove ϵ_i sono gli errori di rappresentazione tali che $|\epsilon_i| < u$. Operando in questo modo, già prima di iniziare i calcoli, abbiamo a che fare con l'errore relativo

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

definito se $f(x) \neq 0$. Tale errore viene chiamato *errore inerente* ed è l'errore dovuto agli errori di rappresentazione. Cioè esso è indotto nella funzione dal fatto che il valore della variabile indipendente $x \in \mathbb{R}^n$ viene alterato in $\tilde{x} \in \mathcal{F}^n$.

4.1 Funzioni razionali

Supponiamo ora che la funzione $f(x) : \mathbb{R} \to \mathbb{R}$ sia razionale, cioè una funzione data dal quoziente di due polinomi, dove Ω è l'insieme dei valori in cui il denominatore non si annulla. Le funzioni razionali sono le sole che si possono calcolare con un numero finito di operazioni aritmetiche. Vogliamo calcolare il valore di $f(\tilde{x})$ eseguendo una opportuna sequenza di operazioni aritmetiche che per semplicità definiremo *algoritmo di calcolo* o più semplicemente algoritmo. Nella realizzazione in aritmetica floating point di un algoritmo ogni operazione aritmetica potenzialmente introduce un errore locale limitato superiormente in valore assoluto dalla precisione di macchina.

In questo modo il valore che otteniamo alla fine dei calcoli in generale non coinciderà con quello di $f(\tilde{x})$ ma sarà qualcosa di diverso in generale che indichiamo con $\varphi(\tilde{x})$. Definiamo quindi l'*errore algoritmico* come

$$\epsilon_{\text{alg}} = \frac{\varphi(\tilde{x}) - f(\tilde{x})}{f(\tilde{x})}.$$

L'errore algoritmico è quindi generato dall'accumularsi degli errori locali relativi a ciascuna operazione aritmetica eseguita in floating point.

Chiamiamo invece `errore totale` la quantità

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

che esprime di quanto il valore effettivamente calcolato $\varphi(\tilde{x})$ si discosta dal valore $f(x)$ che avremmo voluto calcolare.

Si può dimostrare facilmente che vale

$$\epsilon_{\text{tot}} = \epsilon_{\text{in}} + \epsilon_{\text{alg}} + \epsilon_{\text{in}}\epsilon_{\text{alg}} \doteq \epsilon_{\text{in}} + \epsilon_{\text{alg}}, \tag{4}$$

dove col segno $\doteq$ indichiamo l'uguaglianza delle parti lineari negli errori. Di fatto, con l'operazione $\doteq$ manteniamo solamente la parte lineare nell'errore trascurando tutti i termini di ordine quadratico o superiore. Questo tipo di analisi, detta *al primo ordine* è significativa in concreto poiché nella pratica gli errori sono piccoli e i loro prodotti o le loro potenze intere con esponente maggiore di 1 diventano trascurabili.

La dimostrazione di (4) è un semplice conto. Vale infatti

$$\epsilon_{\text{tot}} = \frac{\varphi(\tilde{x})}{f(x)} - 1 = \frac{\varphi(\tilde{x})}{f(\tilde{x})}\frac{f(\tilde{x})}{f(x)} - 1 = (\epsilon_{\text{alg}} + 1)(\epsilon_{\text{in}} + 1) - 1 = \epsilon_{\text{alg}} + \epsilon_{\text{in}} + \epsilon_{\text{alg}}\epsilon_{\text{in}}.$$

Cioè, in una analisi al primo ordine, l'errore totale può essere scisso nella somma dell'errore inerente e di quello algoritmico. Per cui basta studiare separatamente questi due tipi di errori per avere il valore dell'errore totale. È quindi importante disporre di strumenti per studiare l'errore inerente ϵ_{in} e l'errore algoritmico ϵ_{alg}.

4.2 Funzioni non razionali

Nel caso di una funzione non razionale $g(x) : \Omega \subset \mathbb{R} \to \mathbb{R}$, vale ancora la definizione di errore inerente, mentre non possiamo definire un errore algoritmico poiché $g(x)$ non può essere calcolata in un numero finito di operazioni aritmetiche. Per poter calcolare $g(x)$ dobbiamo selezionare una funzione razionale $f(x)$ che ben approssimi $g(x)$. Per questo introduciamo *l'errore analitico* definito da

$$\epsilon_{\text{an}} = \frac{f(x) - g(x)}{g(x)}$$

che esprime di quanto si discosta la funzione razionale $f(x)$ dalla $g(x)$.

Un esempio tipico è il calcolo di e^x, con $x > 0$, mediante la formula

$$e^x = \sum_{i=0}^{+\infty} \frac{x^i}{i!}$$

per cui si può scegliere la funzione razionale

$$f(x) = \sum_{i=0}^{n} \frac{x^i}{i!}$$

dove n è sufficientemente grande in modo che il resto $x^{n+1}e^{\xi}/(n+1)! < x^{n+1}e^x/(n+1)!$, con $0 < \xi < x$, sia minore di ue^x. Questo si ottiene se n è tale che $x^{n+1}/(n+1)! < u$. Infatti, sotto quest'ultima condizione, aggiungere ulteriori addendi alla quantità $\sum_{i=0}^{n} x^i/i!$ non cambia le prime t cifre della rappresentazione in base.

La scelta di $g(x)$ può essere fatta in vari modi ad esempio troncando degli sviluppi in serie, come si è fatto nel calcolo dell'esponenziale, oppure mediante tecniche di interpolazione, approssimanti di Padé ed altro ancora.

Considerando l'errore totale come $(\varphi(\tilde{x}) - g(x))/g(x)$, si può dimostrare la seguente proprietà

$$\epsilon_{\text{tot}} = \epsilon_{\text{in}}+\epsilon_{\text{alg}}+\epsilon_{\text{an}}(\tilde{x})+\epsilon_{\text{in}}\epsilon_{\text{alg}}+\epsilon_{\text{in}}\epsilon_{\text{an}}(\tilde{x})+\epsilon_{\text{alg}}\epsilon_{\text{an}}(\tilde{x})+\epsilon_{\text{alg}}\epsilon_{\text{an}}(\tilde{x})\epsilon_{\text{in}} \doteq \epsilon_{\text{in}}+\epsilon_{\text{alg}}+\epsilon_{\text{an}}(\tilde{x}).$$

Infatti si ha

$$\begin{aligned}
\epsilon_{\text{tot}} &= \frac{\varphi(\tilde{x})}{g(x)} - 1 = \frac{\varphi(\tilde{x})}{f(\tilde{x})}\frac{f(\tilde{x})}{g(\tilde{x})}\frac{g(\tilde{x})}{g(x)} - 1 \\
&= (\epsilon_{\text{alg}} + 1)(\epsilon_{\text{an}}(\tilde{x}) + 1)(\epsilon_{\text{in}} + 1) - 1 \\
&= \epsilon_{\text{alg}} + \epsilon_{\text{an}}(\tilde{x}) + \epsilon_{\text{in}} + \epsilon_{\text{alg}}\epsilon_{\text{in}} + \epsilon_{\text{alg}}\epsilon_{\text{an}}(\tilde{x}) + \epsilon_{\text{an}}(\tilde{x})\epsilon_{\text{in}} + \epsilon_{\text{alg}}\epsilon_{\text{an}}(\tilde{x})\epsilon_{\text{in}}.
\end{aligned}$$

Quindi, ai fini dell'analisi degli errori possiamo studiare separatamente gli errori di ϵ_{in}, ϵ_{alg}, ϵ_{an}. Per quanto riguarda l'errore analitico possiamo usare gli strumenti dell'analisi e della teoria dell'approssimazione di funzioni. Lo studio dell'errore inerente e algoritmico viene riportato di seguito.

4.3 Analisi dell'errore inerente

Se $n = 1$, cioè se $f(x) : \mathbb{R} \to \mathbb{R}$, allora assumendo che $f(x)$ sia definita e derivabile almeno due volte con continuità nel segmento di estremi x e $\tilde{x}$, uno sviluppo in serie di Taylor di $f(x)$ fornisce

$$f(\tilde{x}) = f(x) + (\tilde{x} - x)f'(x) + \frac{1}{2}(\tilde{x} - x)^2 f''(\xi), \quad |\xi - x| < |\xi - \tilde{x}|.$$

Da cui, considerando l'errore di rappresentazione $\delta_x = (\tilde{x} - x)/x$, si ricava

$$\epsilon_{\text{in}} = \delta_x \frac{xf'(x)}{f(x)} + \delta_x^2 \frac{x^2 f''(\xi)}{f(x)} \doteq \delta_x \frac{xf'(x)}{f(x)} \tag{5}$$

La quantità $\frac{xf'(x)}{f(x)}$ che compare nella (5) viene detta *coefficiente di amplificazione* e ci dice di quanto l'errore relativo δ_x presente nei dati viene amplificato (o ridotto) nel valore di $f(x)$ in una analisi al primo ordine. Ad esempio, se $f(x) = x^p$, con p intero, un semplice calcolo mostra che il coefficiente di amplificazione di $f(x)$ è p, cioè un errore relativo δ_x presente nella x viene amplificato

di p volte nella $f(x)$. Mentre per la funzione $x^{1/p}$ il coefficiente di amplificazione è $1/p$, cioè l'errore relativo nella x viene ridotto di p volte nella $f(x)$. Se $f(x) = \log x$ allora il coefficiente di amplificazione è $1/\log x$. Per cui, se $x > e$ o se $x < 1/e$ c'è una riduzione di errore.

Si osservi che la definizione di errore inerente non richiede che $f(x)$ sia razionale, basta solo che sia definita e sufficientemente regolare sull'intervallo di estremi x e $\tilde{x}$.

Nel caso di funzioni $f : \mathbb{R}^n \to \mathbb{R}$ vale una formula analoga per l'errore inerente. Infatti, posto $x = (x_i)$ e $\delta_{x_i} = (\tilde{x}_i - x_i)/x_i$ per $i = 1, \ldots, n$, risulta

$$\epsilon_{\text{in}} \doteq \sum_{i=1}^{n} \delta_{x_i} C_i, \qquad C_i = \frac{x_i \frac{\partial f(x)}{\partial x_i}}{f(x)}. \tag{6}$$

Le quantità C_i sono i coefficienti di amplificazione rispetto alla variabile x_i, per $i = 1, \ldots, n$, dove il simbolo $\partial f(x)/\partial x_i$ denota la derivata di $f(x)$ rispetto alla variabile x_i.

Legato all'errore inerente è il concetto di *condizionamento* di un problema. Si dice che un problema è ben condizionato se una "piccola" variazione relativa dei valori di input produce una "piccola" variazione relativa dei valori di output. Si dice *mal condizionato* se una piccola variazione relativa in input produce una "grande" variazione relativa nell'output. In altri termini, il condizionamento di un problema, quale il calcolo di una funzione, è ben o mal condizionato a seconda della grandezza in modulo dei coefficienti di amplificazione.

4.4 Analisi dell'errore algoritmico

Per l'errore inerente abbiamo introdotto lo strumento dei coefficienti di amplificazione che ci permette, mediante uno studio analitico, di calcolare tale errore in modo agevole. Per studiare l'errore algoritmico occorre faticare un po' di più.

Per studiare l'errore algoritmico consideriamo la più semplice funzione possibile $f(x_1, x_2) = x_1 \,\text{op}\, x_2$, dove op è una delle quattro operazioni aritmetiche, col più semplice algoritmo possibile: quello che esegue una singola operazione aritmetica

$$s = x_1 \,\text{op}\, x_2.$$

Nell'esecuzione di questo semplice algoritmo in aritmetica floating point l'unico errore generato dall'aritmetica approssimata è l'errore locale δ generato dal troncamento del risultato dell'operazione aritmetica. Infatti il valore $\tilde{s}$ effettivamente calcolato è dato da

$$\tilde{s} = (\tilde{x}_1 \,\text{op}\, \tilde{x}_2)(1 + \delta),$$

dove $|\delta| < u$ è l'errore locale, $\tilde{x}_1 = x_1(1 + \epsilon_{x_1})$ e $\tilde{x}_2 = x_2(1 + \epsilon_{x_2})$ sono i valori approssimati degli operandi. Gli errori ϵ_{x_1} e ϵ_{x_2} possono essere gli errori di rappresentazione di x_1 e di x_2, se essi sono dati in input, oppure possono essere gli errori accumulati nelle operazioni precedentemente svolte per calcolare x_1 e

Operazione	C_1	C_2
moltiplicazione	1	1
divisione	1	-1
addizione	$\frac{x_1}{x_1+x_2}$	$\frac{x_2}{x_1+x_2}$
sottrazione	$\frac{x_1}{x_1-x_2}$	$-\frac{x_2}{x_1-x_2}$

Tabella 2: Coefficienti di amplificazione delle operazioni aritmetiche

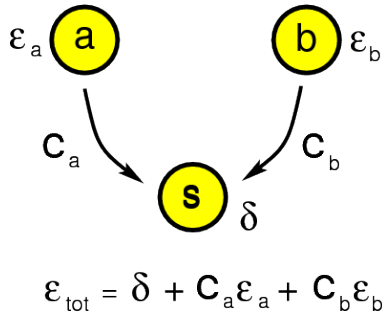

Figura 1: Errore totale in una singola operazione aritmetica

x_2. Quest'ultimo caso è quello che si avrebbe considerando questa singola operazione aritmetica come singola parte di un algoritmo più complesso costituito da più operazioni.

Dalla (4) è evidente che l'errore totale in s, al primo ordine, è dato dalla somma dell'errore algoritmico cioè δ e dell'errore inerente, cioè $C_1\epsilon_{x_1} + C_2\epsilon_{x_2}$, dove C_1 e C_2 sono i coefficienti di amplificazione relativamente a x_1 ed a x_2 della funzione $f(x_1, x_2) = x_1 \,\mathrm{op}\, x_2$, cioè

$$\tilde{s} \doteq (x_1 \,\mathrm{op}\, x_2)(1 + \delta + C_1\epsilon_{x_1} + C_2\epsilon_{x_2}).$$

Diventa quindi determinante studiare i coefficienti di amplificazione delle quattro funzioni $x_1 + x_2$, $x_1 - x_2$, $x_1 \times x_2$, x_1/x_2.

Un semplice calcolo ci permette di ottenere i coefficienti di amplificazione C_1 e C_2 relativi alle due variabili x_1 e x_2 che sono riportati nella tabella 2.

La figura 1 mostra in modo grafico il flusso delle operazioni e degli errori nell'esecuzione di una singola operazione applicata agli operandi a e b.

Si osserva che le operazioni di moltiplicazione e di divisione non amplificano eventuali errori presenti negli operandi. Anche l'addizione tra numeri di segno concorde ha dei coefficienti di amplificazione più piccoli di 1 in valore assoluto.caso L'unica operazione pericolosa che può amplificare in modo incontrollato gli errori presenti nei dati è la sottrazione di numeri concordi o l'addizione di numeri discordi in segno. Infatti in questi casi i rapporti $x_1/(x_1+x_2)$ e $x_2/(x_1+x_2)$ possono assumere valori arbitrariamente grandi in valore assoluto.

Questo fenomeno di amplificazione degli errori che si manifesta nel caso di somma di numeri di segno opposto o sottrazione di numeri di segno concorde viene chiamato *cancellazione numerica*. Il termine è motivato dal fatto che due numeri dello stesso segno che nella sottrazione danno un risultato piccolo in valore assoluto hanno necessariamente molte cifre in comune che si cancellano nell'operazione di sottrazione.

Esempio. Siano $a = 0.12345678$ e $b = 0.12345675$ le rappresentazioni in base 10 di due numeri. Siano $\tilde{a} = 0.12345679$, $\tilde{b} = 0.12345674$ i due valori perturbati in cui $\epsilon_a = 0.81 \times 10^{-7}$, $\epsilon_b = -0.81 \times 10^{-7}$. Risulta $c = a - b = 0.3 \times 10^{-7}$, mentre $\tilde{c} = \tilde{a} - \tilde{b} = 0.5 \times 10^{-7}$, dove la sottrazione è svolta in modo esatto. L'errore che compare nel risultato è $(\tilde{c} - c)/c = 0.4$. Si è passati da un errore relativo in a e in b dell'ordine di 1 su 10 milioni ad un errore relativo nel risultato del 40%. Nello svolgere la sottrazione si verifica la cancellazione di cifre

$$\begin{array}{ll} 0.12345679 & - \\ 0.12345674 & = \\ \hline 0.00000005 & \end{array}$$

È importante ribadire che il fenomeno della cancellazione consiste nella amplificazione degli errori presenti negli operandi e non è dovuta all'errore locale dell'addizione o sottrazione eseguita.

Affinchè un algoritmo non amplifichi troppo l'errore è importante fare in modo che non si presentino delle cancellazioni nei singoli passi dell'algoritmo stesso. Questa semplice regola pratica può essere applicata senza difficoltà in molte situazioni. Abbiamo quindi la seguente ricetta da seguire

Evitare di addizionare numeri di segno opposto o, equivalentemente, di sottrarre numeri dello stesso segno

Esempio. Nel calcolare le radici di un polinomio di secondo grado $ax^2 + bx + c$ viene generalmente usata la formula

$$\frac{-b \pm \sqrt{\Delta}}{2a}, \quad \Delta = b^2 - 4ac.$$

In almeno una delle due operazioni in cui il segno $\pm$ è coinvolto si verifica una cancellazione numerica. Ad esempio, se è $b > 0$, allora il calcolo di $x_1 = -b + \sqrt{\Delta}$ comporta una somma di numeri di segno opposto, mentre il calcolo di $x_2 = -b - \sqrt{\Delta}$ è sicuro. La cancellazione nel calcolo di x_1 può essere evitata utilizzando il fatto che $x_1 x_2 = c/a$ per cui possiamo scrivere $x_1 = c/(a * x_2)$. Quest'ultima formula non comporta cancellazioni numeriche.

Esempio. Nell'approssimare il valore di e^x per un valore assegnato di x possiamo usare lo sviluppo in serie

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$$

e sommare finchè il risultato non cambia più cioè finché il resto dello sviluppo in serie diventa più piccolo in valore assoluto della precisione di macchina per e^x. Se $x > 0$ non si verifica cancellazione, ma se $x < 0$ abbiamo una somma a segni alterni in cui, se $x << -1$ il risultato finale è molto più piccolo rispetto agli addendi. Quindi la formula genera cancellazione. In tal caso possiamo rimuovere il problema scrivendo $e^x = 1/e^{-x}$ e approssimando lo sviluppo in serie di e^{-x} che si riconduce ad una somma di termini positivi.

Esempio. Un altro esempio significativo riguarda il calcolo della somma

$$\sum_{i=1}^{2n} \frac{(-1)^{i-1}}{i} = 1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \cdots$$

Se applichiamo la formula così com'è andiamo a sommare e sottrarre quantità positive e quindi si incorre nella cancellazione. Se invece riscriviamo l'espressione come

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

andiamo ad eseguire solo somme di numeri positivi evitando cancellazione.

Un modo per valutare l'errore algoritmico generato da un algoritmo di calcolo consiste nell'applicare a ciascuna operazione aritmetica l'analisi descritta sopra. Possiamo vedere questo nel caso del calcolo della funzione $a^2 - b^2$ mediante i due seguenti schemi di calcolo

Schema 1
$s_1 = a \times a$
$s_2 = b \times b$
$s_3 = s_1 - s_2$

Schema 2
$s_1 = a + b$
$s_2 = a - b$
$s_3 = s_1 \times s_2$

Il primo algoritmo esegue due moltiplicazioni e una addizione, il secondo una addizione, una sottrazione e una moltiplicazione. I due algoritmi sono rappresentati dai grafi in figura 2.

Denotando con ϵ_i l'errore algoritmico sulla variabile s_i, per il primo algoritmo si ha: $\epsilon_1 = \delta_1$, $\epsilon_2 = \delta_2$, da cui

$$\epsilon_3 \doteq \delta_3 + \frac{a^2}{a^2 - b^2}\delta_1 - \frac{b^2}{a^2 - b^2}\delta_2$$

che ci fornisce la maggiorazione al primo ordine

$$|\epsilon_3| < u(1 + \frac{a^2 + b^2}{|a^2 - b^2|}).$$

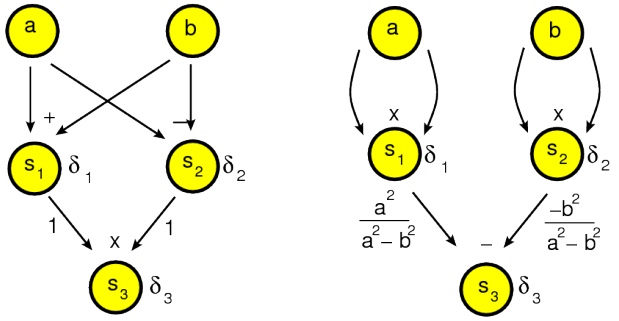

Figura 2: Due algoritmi per il calcolo di $a^2 - b^2$

Nel secondo algoritmo si ha $\epsilon_1 = \delta_1$, $\epsilon_2 = \delta_2$ e quindi

$$\epsilon_3 \doteq \delta_3 + \delta_1 + \delta_2$$

da cui la maggiorazione al primo ordine

$$|\epsilon_3| < 3u.$$

Un modo formalmente diverso, ma sostanzialmente equivalente di condurre una analisi dell'errore consiste nell'applicare la relazione (3) ad ogni operazione. Ad esempio nel caso del secondo algoritmo, la formula $s_3 = (a - b)(a + b)$ si trasforma in

$$\tilde{s}_3 = [(a - b)(1 + \delta_1)][(a + b)(1 + \delta_2)](1 + \delta_3) \doteq (a^2 - b^2)(1 + \delta_1 + \delta_2 + \delta_3).$$

5 Analisi all'indietro (backward analysis)

L'analisi dell'errore che abbiamo descritto ha l'obiettivo di arrivare a dare maggiorazioni al valore assoluto dell'errore algoritmico ottenuto alla fine dei calcoli ed è chiamata *analisi in avanti*. Generalmente una analisi di questo tipo è piuttosto tecnica e laboriosa. Una possibilità diversa che in molti casi semplifica lo studio dell'errore è l'*analisi all'indietro* introdotta da J. H. Wilkinson. La descriviamo prima nel caso di una singola operazione.

Consideriamo la somma $z \in \mathbb{R}$ di due numeri di macchina $x, y \in \mathcal{F}$, cioè $z = x + y$. Se l'operazione viene eseguita in aritmetica floating point si otterrà un valore $\tilde{z} \in \mathcal{F}$ tale che $\tilde{z} = (x + y)(1 + \delta)$ dove $|\delta| < u$. Questa espressione la possiamo scrivere in questa forma

$$\tilde{z} = \hat{x} + \hat{y}, \quad \hat{x} = x(1 + \delta_x) \in \mathbb{R}, \quad \hat{y} = y(1 + \delta_y) \in \mathbb{R}.$$

Cioè il risultato *effettivamente calcolato in aritmetica floating* point lo posso vedere come il risultato *calcolato in modo esatto* a partire però dai valori $\hat{x}$ e $\hat{y}$

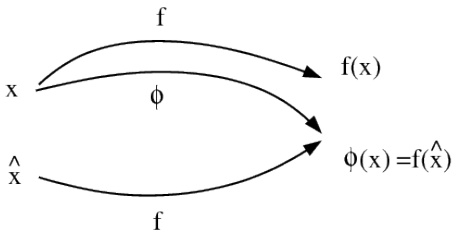

Figura 3: Analisi all'indietro dell'errore

che rispetto ai valori originali x, y hanno un errore relativo rispettivamente δ_x e δ_y.

In generale, sia $f(x_1, \ldots, x_n)$ una funzione razionale e si denoti $\varphi(x_1, \ldots, x_n)$ la funzione definita su $\mathcal{F}^n$ i cui valori sono ottenuti calcolando $f(x_1, \ldots, x_n)$ con l'aritmetica di macchina. Nell'analisi all'indietro dell'errore si cercano delle perturbazioni $\delta_1, \ldots, \delta_n$ tali che denotando con $\hat{x}_i = x_i(1 + \delta_i)$ per $i = 1, \ldots, n$ risulti

$$\varphi(x_1, \ldots, x_n) = f(\hat{x}_1, \ldots, \hat{x}_n).$$

Cioè si cerca di esprimere il valore di una funzione effettivamente calcolato in aritmetica di macchina come il valore della funzione originale $f(x_1, \ldots, x_n)$ calcolato però in un punto $\hat{x} = (\hat{x}_1, \ldots, \hat{x}_n)$ leggermente spostato. Lo scopo è quello di dare delle maggiorazioni al valore assoluto delle perturbazioni δ_i. In questo modo l'errore algoritmico viene visto formalmente come un errore inerente, cioè causato da una perturbazione dell'input. A questo punto, se vogliamo dare maggiorazioni all'errore algoritmico conoscendo limitazioni superiori al valore assoluto delle perturbazioni, possiamo applicare i coefficienti di perturbazione e usare l'espressione (5).

La figura 3 mostra graficamente l'idea alla base della analisi all'indietro dell'errore.

Occorre dire che in generale non è sempre possibile svolgere una analisi all'indietro. Questo accade tipicamente quando il numero di variabili in gioco è inferiore al numero di operazioni da svolgere.

6 Esempi

L'errore totale generato nel calcolo del prodotto di n numeri si analizza facilmente sia mediante una analisi in avanti che mediante una analisi all'indietro. Sia $f(x_1, \ldots, x_n) = \prod_{i=1}^{n} x_i$. Il coefficiente di amplificazione rispetto alla variabile x_i è 1. Quindi l'errore inerente è dato al primo ordine da

$$\epsilon_{\text{in}} \doteq \sum_{i=1}^{n} \epsilon_{x_i},$$

per cui, se $|\epsilon_{x_i}| < u$ allora $|\epsilon_{\text{in}}| < nu$.

Per quanto riguarda l'errore algoritmico, calcolando il prodotto mediante la formula

$$(\cdots((x_1 \times x_2) \times x_3) \times \cdots) \times x_n$$

si ha

$$\varphi = \prod_{i=1}^{n} x_i \prod_{j=1}^{n-1}(1+\delta_j)$$

dove δ_i è l'errore locale dell'i-esima moltiplicazione. Per cui l'errore algoritmico al primo ordine è maggiorato in valore assoluto da $(n-1)u$.

L'analisi all'indietro si effettua scrivendo la relazione precedente come $\prod_{i=1}^{n} \hat{x}_i$ con $\hat{x}_i = x_i(1+\delta_i)$ per $i = 1, \ldots, n-1$, $\hat{x}_n = x_n$.

Una situazione più problematica si incontra nello studio degli errori di una somma di n termini. Infatti, già nell'analisi dell'errore inerente si incontrano coefficienti di amplificazione dati da

$$x_i / \sum_{j=1}^{n} x_j$$

che possono avere valore assoluto arbitrariamente elevato a meno che la somma non abbia tutti addendi dello stesso segno. In questo caso la somma dei valori assoluti dei coefficienti di amplificazione fa 1 per cui l'errore inerente è maggiorato in modulo al primo ordine da u.

Invece per l'errore algoritmico occorre prima specificare in che modo la somma di n addendi viene calcolata. Due tra i numerosi modi diversi, legati alla proprietà associativa dell'addizione, sono dati dal metodo di somma in sequenza e dal metodo di somma in parallelo. Il primo procede secondo lo schema

$s_0 = x_1$
$s_i = s_{i-1} + x_{i+1}$, per $i = 1, \ldots, n-1$

Il secondo procede in base allo schema che per semplicità riportiamo nel caso di $n = 2^p$, p intero positivo.

$s_i^{(0)} = x_i$ per $i = 1, \ldots, n$
$s_i^{(k)} = s_{2i-1}^{(k-1)} + s_{2i}^{(k-1)}$ per $i = 1, \ldots, n/2^k$, $k = 1, 2, \ldots, p-1$.

Se n non fosse potenza intera di 2 basta porre $x_i = 0$ per $i = n+1, \ldots, 2^p$ dove 2^p è la più piccola potenza intera di 2 maggiore o uguale a n.

I grafi relativi al flusso delle operazioni sono riportati nelle figure 4, 5.

Di seguito si riportano i codici *Octave* dei due metodi di somma dove la somma in parallelo viene implementata in modo ricorsivo.

```
function s=somma(x)
% Algoritmo di somma in sequenza
```

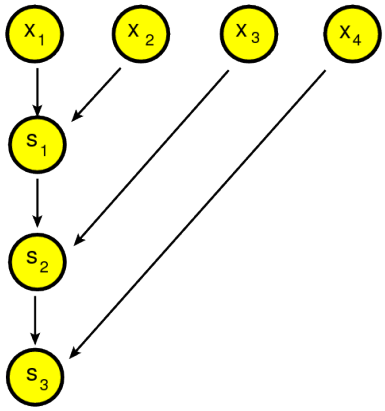

Figura 4: Somma sequenziale

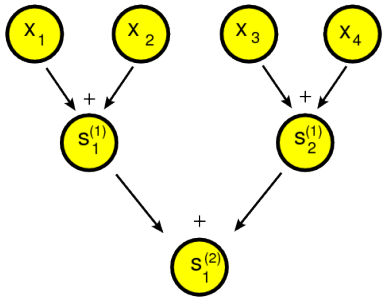

Figura 5: Somma in parallelo

Errori algoritmici	sequenziale	parallelo
crescente	$(n-1)u$	$\lceil \log_2 n \rceil u$
decrescente	$\frac{n}{2}u$	$\lceil \log_2 n \rceil u$

```
  n=length(x);
  s=x(1);
  for i=2:n
    s=s+x(i);
  endfor
endfunction

function s=somma_p(x)
% Algoritmo di somma in parallelo
% implementazione ricorsiva
  n=length(x);
  if n==2
  s=x(1)+x(2);
  elseif n==1
    s=x(1)
  else
    if mod(n,2)==0              % n pari
      y=x(1:2:n)+x(2:2:n);
      s=somma_p(y);
    else                        % n dispari
      y=x(1:2:n-1)+x(2:2:n-1);
      s=somma_p(y)+x(n);
    endif
  endif
endfunction
```

Si può dimostrare che nel caso di coefficienti non negativi l'errore algoritmico generato dai due algoritmi con i due ordinamenti diversi è maggiorato al primo ordine dalle seguenti quantità

Una analisi all'indietro del metodo di somma sequenziale fornisce il seguente risultato

$$\text{float}(\texttt{somma(x)}) = \sum_{i=1}^{n} \tilde{x}_i, \quad \tilde{x}_i = x_i(1+\epsilon_i^{(n)}), \quad |\epsilon_i^{(n)}| < u(n-i+1)$$

Per dimostrare questo si procede per induzione su n. Se $n = 2$ allora $\text{float}(x_1+x_2) = (x_1+x_2)(1+\delta_1) = \tilde{x}_1+\tilde{x}_2$, dove $\tilde{x}_1 = x_1(1+\delta_1)$, $\tilde{x}_2 = x_2(1+\delta_1)$ con $|\delta_1| < u$, e quindi la proprietà è valida. Assumendo valida la proprietà per $n-1$ si considera il caso n. Vale $s_n = s_{n-1} + x_n$, con $s_n = \sum_{i=1}^{n} x_i$. Per cui, il valore effettivamente calcolato $\tilde{s}_n$ è tale che $\tilde{s}_n = (\tilde{s}_{n-1} + x_n)(1+\delta_{n-1})$ con δ_{n-1} errore locale dell'addizione. Ne segue $\tilde{s}_n = \sum_{i=1}^{n-1} x_i(1+\epsilon_i^{(n-1)})(1+\delta_{n-1})$.

Da cui $\epsilon_i^{(n)} \doteq \epsilon_i^{(n-1)} + \delta_{n-1}$ per $i = 1, \ldots, n-1$, $\epsilon_n^{(n)} = \delta_{n-1}$. Quindi $|\epsilon_i^{(n)}| < |\epsilon_i^{(n-1)}| + u < n - i + 1$. La dimostrazione è completa.

Una analisi all'indietro del metodo di somma parallela fornisce il seguente risultato

$$\text{float}(\texttt{somma\_p(x)}) = \sum_{i=1}^{n} \tilde{x}_i, \quad \tilde{x}_i = x_i(1+\epsilon_i), \quad |\epsilon_i| < u\lceil \log_2 n \rceil.$$

Per semplicità dimostriamo questo fatto nel caso in cui $n = 2^q$ con q intero positivo. Nel caso generale basta aggiungere addendi nulli fino ad arrivare ad un numero di addendi uguale alla prima potenza di 2 maggiore o uguale ad n. Riscriviamo l'algoritmo nel seguente modo:

$$\begin{aligned} &s_i^{(0)} = x_i, \quad i = 1, \ldots, n \\ &s_i^{(k+1)} = s_{2i-1}^{(k)} + s_{2i}^{(k)}, \quad i = 1, \ldots, n/2^k, \quad k = 0, 1, \ldots, p-1 \end{aligned}$$

Dimostriamo per induzione su k che $\tilde{s}_i^{(k)} = s_i^{(k)}(1+\epsilon_i^{(k)})$, $|\epsilon_i^{(k)}| < ku$. Per $k = 0$ la relazione è chiaramente verificata. Per il passo induttivo si ha

$$\tilde{s}_i^{(k+1)} = (\tilde{s}_{2i-1}^{(k)} + \tilde{s}_{2i}^{(k)})(1+\delta_i^{(k)}), \quad |\delta_i^{(k)}| < u.$$

Da cui

$$\tilde{s}_i^{(k+1)} \doteq s_{2i-1}^{(k)}(1+\epsilon_{2i-1}^{(k)} + \delta_i^{(k)}) + s_{2i}^{(k)}(1+\epsilon_{2i}^{(k)} + \delta_i^{(k)}).$$

Dall'ipotesi induttiva si deduce che $|\epsilon_{2i-1}^{(k+1)}| = |\epsilon_{2i-1}^{(k)} + \delta_i^{(k)}| < (k+1)u$ e, similmente, $|\epsilon_{2i}^{(k+1)}| = |\epsilon_{2i}^{(k)} + \delta_i^{(k)}| < (k+1)u$. Questo completa la dimostrazione.

Un esempio significativo di analisi all'indietro riguarda il calcolo del determinante di una matrice tridiagonale $n \times n$

$$A_n = \begin{bmatrix} a_1 & b_1 & & & \\ c_2 & a_2 & b_2 & & \\ & \ddots & \ddots & \ddots & \\ & & c_{n-1} & a_{n-1} & b_{n-1} \\ & & & c_n & a_n \end{bmatrix}$$

Infatti, denotando con $x_n = \det A_n$, calcolando il determinante con regola di Laplace dello sviluppo per righe si ha

$$\begin{aligned} &x_n = a_n x_{n-1} - c_n b_{n-1} x_{n-2} \\ &x_1 = a_1, \\ &x_0 = 1. \end{aligned}$$

I valori $\tilde{x}_i$ effettivamente calcolati verificano la relazione

$$\begin{aligned} &\tilde{x}_n = a_n \tilde{x}_{n-1}(1+\alpha_n)(1+\beta_n) - c_n b_{n-1} \tilde{x}_{n-2}(1+\beta_n)(1+\gamma_n)(1+\delta_n) \\ &\tilde{x}_1 = a_1, \\ &\tilde{x}_0 = 1. \end{aligned}$$

dove $\alpha_n, \beta_n, \gamma_n$ e δ_n sono gli errori locali generati nelle quattro operazioni aritmetiche e sono maggiorati in valore assoluto dalla precisione di macchina u. Per cui, definendo $\tilde{a}_n = a_n(1+\alpha_n)(1+\beta_n)$ e $\tilde{b}_{n-1} = b_{n-1}(1+\delta_n)$, $\tilde{c}_n = c_n(1+\beta_n)(1+\gamma_n)$, si ottiene

$$\begin{aligned}
&\tilde{x}_n = \tilde{a}_n\tilde{x}_{n-1} - \tilde{c}_n\tilde{b}_{n-1}\tilde{x}_{n-1}\\
&\tilde{x}_1 = a_1,\\
&\tilde{x}_0 = 1.
\end{aligned}$$

cioè i valori effettivamente calcolati sono i determinanti delle matrici tridiagonali definiti da $\tilde{a}_i$, $\tilde{b}_i$ e $\tilde{c}_i$. Inoltre, in una analisi al primo ordine, le perturbazioni relative indotte nelle variabili di input a_n, b_{n-1} e c_n sono limitati rispettivamente da $2u, u$ e $2u$. Si ha quindi un algoritmo stabile all'indietro.

7 Esercizi

In questo paragrafo abbiamo raccolto esercizi relativi all'analisi degli errori alcuni dei quali riportano la risoluzione. Spesso useremo il simbolo $\dot{\leq}$ per denotare la diseguaglianza a meno di termini di ordine u^2 o superiore.

Un modo sistematico per trattare l'analisi all'indietro, che conviene usare quando una analisi più diretta diventa difficoltosa, è operare nel seguente modo. Supponiamo per semplicità di dover valutare una funzione di 3 variabili $f(x, y, z)$ denotiamo $\varphi(x, y, z)$ la funzione effettivamente calcolata in aritmetica floating point ed esprimiamola in termini degli errori locali sostituendo ogni operazione aritmetica del tipo $a \operatorname{op} b$ con l'operazione di macchina $(a \operatorname{op} b)(1 + \epsilon)$ dove ϵ è l'errore locale tale che $|\epsilon| < u$. Denotiamo poi $\hat{x} = x(1 + \delta_x)$, $\hat{y} = y(1 + \delta_y)$, $\hat{z} = z(1 + \delta_z)$ i valori perturbati rispettivamente di x, y, z. Per ricavare le tre perturbazioni incognite δ_x, δ_y, δ_z si scrive il sistema lineare ottenuto uguagliando le parti lineari negli errori di $f(\hat{x}, \hat{y}, \hat{z})$ e di $\varphi(x, y, z)$. Se il sistema lineare ha soluzione allora l'analisi all'indietro può essere completata risolvendo il sistema.

Un esempio di questo modo di procedere è dato nell'esercizio 1 che viene affrontato nei due modi descritti.

Per ottenere stime dell'errore algoritmico, dopo aver svolto l'analisi all'indietro, basta esprimere l'errore algoritmico in funzione delle perturbazioni calcolate e dei corrispondenti coefficienti di amplificazione. Questo è mostrato nell'esercizio 22.

Per condurre l'analisi in avanti è possibile rappresentare l'algoritmo in termini del suo grafo associato e applicare ad ogni nodo la proprietà che l'errore presente nella variabile associata al nodo è dato al primo ordine dalla somma dell'errore locale e dell'errore proveniente dagli operandi moltiplicato per i coefficienti di amplificazione.

Un altro modo equivalente di procedere è sostituire ad ogni operazione del tipo $a \operatorname{op} b$ l'operazione di macchina $(a \operatorname{op} b)(1+\epsilon)$, raccogliere la parte che contiene gli errori locali e dividerla per il valore della funzione. Occorre poi maggiorare il valore assoluto di questa quantità usando la diseguaglianza triangolare.

Esercizio 1 Dati numeri di macchina x_1, x_2, x_3, costruire un algoritmo numericamente stabile all'indietro per il calcolo di

$$f(x_1, x_2, x_3) = x_1x_2 + x_2x_3 + x_3x_1.$$

Detto $\varphi(x_1, x_2, x_3)$ il risultato fornito dall'algoritmo in aritmetica floating point, si dimostri che esistono δ_1, δ_2, δ_3 tali che $\varphi(x_1, x_2, x_3) = f(\hat{x}_1, \hat{x}_2, \hat{x}_3)$, $\hat{x}_i = x_i(1 + \delta_i)$, per $i = 1, 2, 3$. Dare maggiorazioni a $|\delta_i|$, $i = 1, 2, 3$.

Soluzione. L'algoritmo, che si basa sulla formula

$$f(x_1, x_2, x_3) = (x_1 + x_3)x_2 + (x_1x_3),$$

consiste nell'effettuare le seguenti operazioni:

$$\begin{aligned} s_1 &= x_3 \cdot x_1, \\ s_2 &= x_1 + x_3, \\ s_3 &= s_2 \cdot x_2, \\ s_4 &= s_1 + s_3, \end{aligned}$$

dove $f(x_1, x_2, x_3) = s_4$. Se $\tilde{s}_i$, $i = 1, \ldots, 4$, sono i valori effettivamente calcolati, otteniamo che

$$\begin{aligned} \tilde{s}_1 &= (x_3x_1)(1 + \epsilon_1), \\ \tilde{s}_2 &= (x_1 + x_3)(1 + \epsilon_2), \\ \tilde{s}_3 &= \tilde{s}_2x_2(1 + \epsilon_3), \\ \tilde{s}_4 &= (\tilde{s}_1 + \tilde{s}_3)(1 + \epsilon_4), \end{aligned}$$

dove $|\epsilon_i| < u$, $i = 1, \ldots, 4$ sono gli errori locali generati nelle singole operazioni aritmetiche floating point per il calcolo di s_i. Dunque si ha

$$\varphi(x_1, x_2, x_3) = \big(x_2(x_1 + x_3)(1 + \epsilon_2)(1 + \epsilon_3) + x_3x_1(1 + \epsilon_1)\big)(1 + \epsilon_4) = x_2(x_1 + x_3)(1 + \epsilon_2)(1 + \epsilon_3)(1 + \epsilon_4) + x_3x_1(1 + \epsilon_1)(1 + \epsilon_4).$$

Se usiamo l'approccio sistematico abbiamo che, posto $\hat{x}_1 = x_1(1 + \delta_1)$, $\hat{x}_2 = x_2(1 + \delta_2)$, $\hat{x}_3 = x_3(1 + \delta_3)$, vale

$$f(\hat{x}_1, \hat{x}_2, \hat{x}_3) \doteq x_1x_2(1 + \delta_1 + \delta_2) + x_2x_3(1 + \delta_2 + \delta_3) + x_3x_1(1 + \delta_3 + \delta_1).$$

per cui, dalla relazione ottenuta prendendo le parti lineari negli errori in $\varphi(x_1, x_2, x_3) = f(\hat{x}_1, \hat{x}_2, \hat{x}_3)$ si ottiene il sistema

$$\begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} \delta_1 \\ \delta_2 \\ \delta_3 \end{bmatrix} = \begin{bmatrix} \epsilon_2 + \epsilon_3 + \epsilon_4 \\ \epsilon_2 + \epsilon_3 + \epsilon_4 \\ \epsilon_1 + \epsilon_4 \end{bmatrix}$$

La soluzione del sistema è data da $\delta_3 = (\epsilon_1 + \epsilon_4)/2$, $\delta_2 = \epsilon_2 + \epsilon_3 + (\epsilon_4 - \epsilon_1)/2$, $\delta_1 = (\epsilon_1 + \epsilon_4)/2$.

Invece seguendo un approccio più diretto, poiché

$$(1 + \epsilon_1)(1 + \epsilon_4) \doteq (1 + \epsilon_1 + \epsilon_4) \doteq \left(1 + \frac{\epsilon_1 + \epsilon_4}{2}\right)\left(1 + \frac{\epsilon_1 + \epsilon_4}{2}\right),$$

vale

$$x_3x_1(1+\epsilon_1)(1+\epsilon_4) = \hat{x}_3\hat{x}_1$$

dove $\hat{x}_1 = x_1(1+\delta_1)$, $\hat{x}_3 = x_3(1+\delta_3)$, dove $\delta_1 = \delta_3 \doteq \frac{\epsilon_1+\epsilon_4}{2}$. D'altra parte

$$\begin{array}{l}(1+\epsilon_2)(1+\epsilon_3)(1+\epsilon_4) \doteq (1+\epsilon_2+\epsilon_3+\epsilon_4) = \\ (1+\epsilon_2+\epsilon_3+\epsilon_4)\left(1+\frac{\epsilon_1+\epsilon_4}{2}\right)^{-1}\left(1+\frac{\epsilon_1+\epsilon_4}{2}\right) \doteq \\ (1+\epsilon_2+\epsilon_3+\epsilon_4)\left(1-\frac{\epsilon_1+\epsilon_4}{2}\right)\left(1+\frac{\epsilon_1+\epsilon_4}{2}\right) \doteq \\ \left(1-\frac{\epsilon_1}{2}+\epsilon_2+\epsilon_3+\frac{\epsilon_4}{2}\right)\left(1+\frac{\epsilon_1+\epsilon_4}{2}\right).\end{array}$$

Dunque

$$x_2(x_1+x_3)(1+\epsilon_2)(1+\epsilon_3)(1+\epsilon_4) = \hat{x}_2(\hat{x}_1+\hat{x}_3)$$

dove $\hat{x}_2 = x_2(1+\delta_2)$ e $\delta_2 \doteq -\frac{\epsilon_1}{2}+\epsilon_2+\epsilon_3+\frac{\epsilon_4}{2}$. Quindi $\varphi(x_1,x_2,x_3) = f(\hat{x}_1,\hat{x}_2,\hat{x}_3)$ dove $\hat{x}_i = x_i(1+\delta_i)$ e $|\delta_1| \dot{\leq} \frac{|\epsilon_1|+|\epsilon_4|}{2} < u$, $|\delta_2| \dot{\leq} \frac{|\epsilon_1|}{2}+|\epsilon_2|+|\epsilon_3|+\frac{|\epsilon_4|}{2} < 3u$, $|\delta_3| \dot{\leq} \frac{|\epsilon_1|+|\epsilon_4|}{2} < u$. □

Esercizio 2 Dati numeri di macchina x_1, x_2, costruire un algoritmo che calcoli

$$f(x_1,x_2) = x_1^2/x_2 + x_2/x_1$$

con tre operazioni aritmetiche. Si provi la stabilità all'indietro dimostrando che esistono δ_1, δ_2 tali che $\varphi(x_1,x_2) = f(\tilde{x}_1,\tilde{x}_2)$, $\tilde{x}_i = x_i(1+\delta_i)$, per $i = 1,2$, dove $\varphi(x_1,x_2)$ è il risultato fornito dall'algoritmo in aritmetica floating point. Dare maggiorazioni a $|\delta_i|$, $i = 1,2$.

Soluzione. L'algoritmo si basa sulla formula

$$f(x_1,x_2) = x_1/(x_2/x_1) + (x_2/x_1)$$

ed è dato da

$$\begin{aligned} s_1 &= x_2/x_1, \\ s_2 &= x_1/s_1, \\ s_3 &= s_2+s_1, \end{aligned}$$

dove $s_3 = f(x_1,x_2)$. Operando in aritmetica floating point si ha

$$\begin{aligned} \tilde{s}_1 &= (1+\epsilon)x_2/x_1, \\ \tilde{s}_2 &= (1+\theta)x_1/\tilde{s}_1, \\ \tilde{s}_3 &= (\tilde{s}_2+\tilde{s}_1)(1+\eta). \end{aligned}$$

Dove abbiamo indicato con ϵ, θ, η gli errori locali generati dalle singole operazioni aritmetiche. Risulta quindi

$$\tilde{s}_3 = \frac{(1+\eta)(1+\theta)x_1}{(x_2/x_1)(1+\epsilon)} + (x_2/x_1)(1+\epsilon)(1+\eta)$$

Vale allora

$$\tilde{s}_3 = \tilde{x}_1/(\tilde{x}_2/\tilde{x}_1) + (\tilde{x}_2/\tilde{x}_1)$$

dove si è posto $\tilde{x}_1 = x_1(1+\eta)^2(1+\theta)$, $\tilde{x}_2 = x_2(1+\eta)^3(1+\theta)(1+\epsilon)$. Per cui, essendo gli errori locali maggiorati in modulo dalla precisione di macchina u risulta $|\delta_1| \dot{\leq} 3u$, $|\delta_2| \dot{\leq} 5u$. □

Esercizio 3 Dati numeri di macchina $x_1, x_2 \neq 0$ costruire un algoritmo che calcoli

$$f(x_1, x_2) = x_1^2/x_2 + x_2^2/x_1$$

con 4 operazioni aritmetiche. Svolgere l'analisi in avanti dell'errore e dare maggiorazioni al valore assoluto dell'errore algoritmico. Si studi in particolare il caso in cui $x_1 x_2 > 0$.

Soluzione. L'algoritmo si basa sulla formula

$$f(x_1, x_2) = x_1(x_1/x_2) + x_2/(x_1/x_2)$$

e consiste nell'effettuare le seguenti operazioni:

$$\begin{aligned} s_1 &= x_1/x_2, \\ s_2 &= x_1 s_1, \\ s_3 &= x_2/s_1, \\ s_4 &= s_2 + s_3. \end{aligned}$$

Indichiamo con ϵ_i l'errore algoritmico per il calcolo di s_i e δ_i l'errore locale dovuto alla operazione aritmetica svolta nel calcolo di s_i. Vale allora $\epsilon_1 = \delta_1$ con $|\delta_1| < u$.

Vale inoltre

$$\begin{aligned} \epsilon_2 &\doteq \delta_2 + \epsilon_1, \\ \epsilon_3 &\doteq \delta_3 - \epsilon_1, \\ \epsilon_4 &\doteq \delta_4 + \tfrac{s_2}{s_2+s_3}\epsilon_2 + \tfrac{s_3}{s_2+s_3}\epsilon_3, \end{aligned}$$

con $|\delta_i| < u$, $i = 2, 3, 4$. Dunque

$$\epsilon_4 \doteq \delta_4 + \frac{x_1^2/x_2}{x_1^2/x_2 + x_2^2/x_1}(\delta_1 + \delta_2) + \frac{x_2^2/x_1}{x_1^2/x_2 + x_2^2/x_1}(\delta_3 - \delta_1)$$

da cui

$$|\epsilon_4| \dot{\leq} u\left(1 + 2\frac{|x_1^2/x_2|}{|x_1^2/x_2 + x_2^2/x_1|} + 2\frac{|x_2^2/x_1|}{|x_1^2/x_2 + x_2^2/x_1|}\right).$$

Nel caso $x_1 x_2 > 0$ si ottiene $|\epsilon_4| \dot{\leq} 3u$.

Esercizio 4 Sia $f(x, y) = xy + (x^2)/y = x(y + x/y)$. Si effettui l'analisi in avanti dei due algoritmi per il calcolo di $f(x, y)$ definiti dalle espressioni precedenti. Si confrontino le prestazioni dei due metodi in termini di complessità e di stabilità numerica. Si valuti anche l'errore totale.

Soluzione. In una analisi al primo ordine l'errore totale è, la somma dell'errore inerente e di quello algoritmico. L'errore inerente è dato da $\epsilon_{in} \doteq c_1\sigma_1 + c_2\sigma_2$ dove $|\sigma_i| < u$ per $i = 1, 2$ e $c_1 = \frac{x}{f(x,y)}\frac{\partial f}{\partial x} = \frac{2x+y^2}{y^2+x}$, $c_2 = \frac{y}{f(x,y)}\frac{\partial f}{\partial y} = \frac{y^2-x}{y^2+x}$. Dunque

$$\epsilon_{in} \ \dot{\leq} \ u\left(\frac{|2x+y^2|}{|y^2+x|} + \frac{|y^2-x|}{|y^2+x|}\right).$$

Per l'analisi dell'errore algoritmico supponiamo che x e y siano numeri di macchina.

Il primo algoritmo consiste nell'effettuare le seguenti operazioni:

$$\begin{aligned} s_1 &= xy, \\ s_2 &= x^2, \\ s_3 &= s_2/y, \\ s_4 &= s_1 + s_3. \end{aligned}$$

Il numero di operazioni è 4. Indichiamo con ϵ_i l'errore algoritmico per il calcolo di s_i e δ_i l'errore locale generato nel calcolo di s_i. Vale $\epsilon_1 = \delta_1$ e $\epsilon_2 = \delta_2$, con $|\delta_i| < u$ per $i = 1, 2$; inoltre

$$\begin{aligned} \epsilon_3 &\doteq \delta_3 + \epsilon_2 \\ \epsilon_4 &\doteq \delta_4 + \tfrac{s_1}{s_1+s_3}\epsilon_1 + \tfrac{s_3}{s_1+s_3}\epsilon_3 \end{aligned}$$

con $|\delta_i| < u$, $i = 3, 4$. Dunque, svolgendo i conti,

$$\epsilon_4 \doteq \delta_4 + \frac{y^2}{x+y^2}\delta_1 + \frac{x}{x+y^2}(\delta_3 + \delta_2),$$

da cui

$$|\epsilon_4| \ \dot{\leq} \ u\left(1 + \frac{y^2 + 2|x|}{|x+y^2|}\right).$$

Nel caso $x > 0$ si ottiene $|\epsilon_4| \ \dot{\leq} \ 3u$.

Il secondo algoritmo consiste nell'effettuare le seguenti operazioni:

$$\begin{aligned} s_1 &= x/y, \\ s_2 &= s_1 + y, \\ s_3 &= xs_2. \end{aligned}$$

Il numero di operazioni è 3. Indichiamo con ϵ_i l'errore algoritmico per il calcolo di s_i e con δ_i l'errore locale generato dall'operazione aritmetica svolta nel calcolare s_i. Vale dunque $\epsilon_1 = \delta_1$ con $|\delta_1| < u$; inoltre

$$\begin{aligned} \epsilon_2 &\doteq \delta_2 + \tfrac{s_1}{s_1+y}\epsilon_1 \\ \epsilon_3 &\doteq \delta_3 + \epsilon_2 \end{aligned}$$

con $|\delta_i| < u$, $i = 2, 3$.

Dunque, svolgendo i conti,

$$\epsilon_3 \doteq \delta_3 + \delta_2 + \frac{x}{x+y^2}\delta_1,$$

da cui

$$|\epsilon_3| \stackrel{\cdot}{\leq} u\left(2+\frac{|x|}{|x+y^2|}\right).$$

Nel caso $x>0$ si ottiene $|\epsilon_3| \stackrel{\cdot}{\leq} 3u$. □

Esercizio 5 Sia $g(x,a,b)=ax+b/x$ e si indichino con $\tilde{g}_1,\ \tilde{g}_2$ i valori ottenuti calcolando $g(x,a,b)$ con l'algoritmo

$$\begin{aligned} s_1 &= a\cdot x,\\ s_2 &= b/x,\\ g_1 &= s_1+s_2, \end{aligned}$$

e con l'algoritmo

$$\begin{aligned} t_1 &= x\cdot x,\\ t_2 &= a\cdot t_1,\\ t_3 &= t_2+b,\\ g_2 &= t_3/x, \end{aligned}$$

Dati $a,b,x\in\mathcal{F}$ dire se esistono valori $\hat{a},\hat{b},\tilde{a},\tilde{b}\in\mathbb{R}$ tali che $\tilde{g}_1=g(x,\hat{a},\hat{b})$, $\tilde{g}_2=g(x,\tilde{a},\tilde{b})$. Nel caso stimare il valore assoluto degli errori relativi $\delta_a,\delta_b,\epsilon_a,\epsilon_b$ tali che $\hat{a}=a(1+\delta_a)$, $\hat{b}=b(1+\delta_b)$, $\tilde{a}=a(1+\epsilon_a)$, $\tilde{b}=b(1+\epsilon_b)$.

Soluzione. Con il primo algoritmo, se $\tilde{s}_1$ e $\tilde{s}_2$ sono i valori effettivamente calcolati, allora $\tilde{s}_1=(ax)(1+\epsilon_1)$ e $\tilde{s}_2=(b/x)(1+\epsilon_2)$ con $|\epsilon_1|,|\epsilon_2|<u$ errori locali. Dunque il valore della funzione effettivamente calcolato sarà

$$\tilde{g}_1=(\tilde{s}_1+\tilde{s}_2)(1+\epsilon_3)=(ax)(1+\epsilon_1)(1+\epsilon_3)+(b/x)(1+\epsilon_2)(1+\epsilon_3),$$

con $|\epsilon_3|<u$. Quindi $\tilde{g}_1=g(x,\hat{a},\hat{b})$ dove $\hat{a}=a(1+\delta_a)$, $\hat{b}=b(1+\delta_b)$, e $|\delta_a|\doteq|\epsilon_1+\epsilon_3|<2u$, $|\delta_b|\doteq|\epsilon_2+\epsilon_3|<2u$.

Con il secondo algoritmo, se $\tilde{t}_i$ sono i valori effettivamente calcolati, allora $\tilde{t}_1=x^2(1+\epsilon_1)$, $\tilde{t}_2=(a\tilde{t}_1)(1+\epsilon_2)$, $\tilde{t}_3=(\tilde{t}_2+b)(1+\epsilon_3)$ dove gli errori locali sono tali che $|\epsilon_1|,|\epsilon_2|,|\epsilon_3|<u$. Dunque il valore della funzione effettivamente calcolato sarà

$$\tilde{g}_2=(\tilde{t}_3/x)(1+\epsilon_4)=(ax(1+\epsilon_1)(1+\epsilon_2)+b/x)(1+\epsilon_3)(1+\epsilon_4),$$

con $|\epsilon_4|<u$. Quindi $\tilde{g}_2=g(x,\hat{a},\hat{b})$ dove $\hat{a}=a(1+\delta_a)$, $\hat{b}=b(1+\delta_b)$, e $|\delta_a|\doteq|\epsilon_1+\epsilon_2+\epsilon_3+\epsilon_4|<4u$, $|\delta_b|\doteq|\epsilon_3+\epsilon_4|<2u$. □

Esercizio 6 Si descriva un algoritmo per il calcolo di $\sum_{i=0}^{2^k-1}x^i$, dati x e k, basato sulla seguente identità

$$\sum_{i=0}^{2^k-1}x^i=(1+x)(1+x^2)(1+x^{2^2})\cdots(1+x^{2^{k-1}})$$

che impieghi al più $3k$ operazioni aritmetiche. Si scriva una function nella sintassi di Octave che lo implementa. Si dia una maggiorazione al primo ordine del valore assoluto dell'errore algoritmico nel caso in cui $0 < x < 1$.

Esercizio 7 Per calcolare la funzione $f(x) = x^3 - 1$, per valori di $x \in \mathcal{F}$, si consideri l'algoritmo $\mathcal{A}_1$ che calcola nell'ordine

$$\begin{aligned} s_1 &= x \cdot x, \\ s_2 &= x \cdot s_1, \\ s_3 &= s_2 - 1, \end{aligned}$$

dove $f(x) = s_3$, e l'algoritmo $\mathcal{A}_2$ che si basa sull'identità $f(x) = (x-1)((x+1)^2 - x)$ e che calcola nell'ordine

$$\begin{aligned} t_1 &= x + 1, \\ t_2 &= t_1 \cdot t_1, \\ t_3 &= t_2 - x, \\ t_4 &= x - 1, \\ t_5 &= t_3 \cdot t_4, \end{aligned}$$

dove $f(x) = t_5$.

a) Mediante un'analisi in avanti si diano maggiorazioni al primo ordine α_1 e α_2 ai valori assoluti degli errori algoritmici generati rispettivamente da $\mathcal{A}_1$ e $\mathcal{A}_2$.

b) Si dimostri che α_2 è limitato superiormente da una costante e che α_1 può assumere valori arbitrariamente grandi.

Soluzione. Consideriamo l'algoritmo $\mathcal{A}_1$. Indichiamo con ϵ_i l'errore algoritmico per il calcolo di s_i e δ_i l'errore locale dovuto all'operazione aritmetica svolta nel calcolo di s_i. Vale allora

$$\begin{aligned} &\epsilon_1 = \delta_1, \\ &\epsilon_2 \doteq \delta_2 + \epsilon_1, \\ &\epsilon_3 \doteq \delta_3 + \tfrac{s_2}{s_3}\epsilon_2, \end{aligned}$$

con $|\delta_i| < u$, $i = 1, 2, 3$, da cui otteniamo

$$\epsilon_3 \doteq \delta_3 + \frac{x^3}{x^3 - 1}(\delta_1 + \delta_2).$$

Passando ai moduli abbiamo

$$|\epsilon_3| \stackrel{\cdot}{\leq} \left(1 + 2\frac{|x^3|}{|x^3 - 1|}\right) u = \alpha_1(x)u,$$

quindi $\alpha_1(x)$ può essere arbitrariamente grande quando x si avvicina a 1.

Consideriamo ora l'algoritmo $\mathcal{A}_2$. Indichiamo con τ_i l'errore algoritmico per il calcolo di t_i e σ_i l'errore locale dovuto alla operazione aritmetica svolta nel calcolo di t_i. Vale allora

$$\begin{aligned}
&\tau_1 = \sigma_1,\\
&\tau_2 \doteq \sigma_2 + \tau_1 + \tau_1,\\
&\tau_3 \doteq \sigma_3 + \tfrac{t_2}{t_3}\tau_2,\\
&\tau_4 = \sigma_4,\\
&\tau_5 \doteq \sigma_5 + \tau_3 + \tau_4,
\end{aligned}$$

con $|\sigma_i| < u$, $i = 1, \ldots, 5$, da cui otteniamo

$$\tau_5 \doteq \sigma_3 + \sigma_4 + \sigma_5 + \frac{(x+1)^2}{x^2+x+1}(2\sigma_1 + \sigma_2).$$

Passando ai moduli abbiamo

$$|\tau_5| \;\dot{\leq}\; 3\left(1 + \frac{(x+1)^2}{x^2+x+1}\right)u = \alpha_2(x)u,$$

e la funzione α_2 è limitata superiormente da una costante.

Osservazione: Se x fosse un numero di macchina, per cui l'errore inerente sarebbe nullo, ci sarebbe un chiaro vantaggio del secondo algoritmo sul primo poiché l'errore algoritmico ha una maggiorazione migliore rispetto al primo. Se peró x non fosse un numero di macchina, allora non ci sarebbe vantaggio di un algoritmo rispetto all'altro. Infatti l'errore inerente sarebbe $\epsilon_{in} = \epsilon\frac{x(3x^2)}{x^3-1}$, dove ϵ è l'errore di rappresentazione. Dunque l'errore inerente è arbitrariamente grande quando x si avvicina a 1 e dominerebbe comunque sull'errore algoritmico.

Esercizio 8 Per calcolare l'espressione $f(a,b,c,d) = (a^2 + bc)/(c + d)$ si consideri l'algoritmo definito da:

$$\begin{aligned}
s_1 &= a \cdot a,\\
s_2 &= b \cdot c,\\
s_3 &= c + d,\\
s_4 &= s_1 + s_2,\\
s_5 &= s_4/s_3,
\end{aligned}$$

dove $f(a,b,c,d) = s_5$. Si provi che tale algoritmo è numericamente stabile all'indietro nei punti in cui $f(a,b,c,d)$ è definita, dimostrando che il valore $\varphi(a,b,c,d)$ effettivamente calcolato in aritmetica floating point è uguale a $f(\tilde{a},\tilde{b},\tilde{c},\tilde{d})$ per opportuni valori $\tilde{a},\tilde{b},\tilde{c},\tilde{d}$. Si diano limitazioni superiori alle perturbazioni $|\tilde{a} - a|, |\tilde{b} - b|, |\tilde{c} - c|, |\tilde{d} - d|$.

Soluzione. Indichiamo con $\tilde{s}_i$ i valori effettivamente calcolati di s_i. Allora

$$\begin{aligned}
\tilde{s}_1 &= a \cdot a(1 + \epsilon_1),\\
\tilde{s}_2 &= b \cdot c(1 + \epsilon_2),\\
\tilde{s}_3 &= (c + d)(1 + \epsilon_3),\\
\tilde{s}_4 &= (\tilde{s}_1 + \tilde{s}_2)(1 + \epsilon_4),\\
s_5 &= \tilde{s}_4/\tilde{s}_3(1 + \epsilon_5),
\end{aligned}$$

con $|\epsilon_i| < u$. Allora, trascurando i termini quadratici negli ϵ_i, vale

$$\tilde{s}_5 \doteq \frac{a^2(1+\epsilon_1+\epsilon_4+\epsilon_5)+bc(1+\epsilon_2+\epsilon_4+\epsilon_5)}{(c+d)(1+\epsilon_3)} \doteq \frac{\hat{a}^2-\hat{b}\hat{c}}{\hat{c}\hat{d}}$$

dove

$$\begin{aligned}
&\hat{a} = a(1+\delta_a), \quad \delta_a = (\epsilon_1+\epsilon_4+\epsilon_5)/2,\\
&\hat{b} = b(1+\delta_b), \quad \delta_b = \epsilon_2+\epsilon_4+\epsilon_5-\epsilon_3,\\
&\hat{c} = c(1+\delta_c), \quad \delta_c = \epsilon_3,\\
&\hat{d} = d(1+\delta_d), \quad \delta_d = \epsilon_3.
\end{aligned}$$

Esercizio 9 Per calcolare la funzione $f(a,b) = a^2+b^2/a$ si consideri l'algoritmo che svolge i seguenti passi:

$$\begin{aligned}
s_1 &= a \cdot a,\\
s_2 &= b \cdot b,\\
s_3 &= s_2/a,\\
s_4 &= s_1 + s_3.
\end{aligned}$$

Si dimostri la stabilità all'indietro dell'algoritmo e si diano maggiorazioni al primo ordine per $|\epsilon_a|, |\epsilon_b|$ tali che $\varphi(a,b) = f(a(1+\epsilon_a), b(1+\epsilon_b))$, dove $\varphi(a,b)$ è il valore ottenuto eseguendo l'algoritmo in aritmetica floating point con numeri di macchina a, b. Si ricavi una maggiorazione al primo ordine del valore assoluto dell'errore algoritmico nel caso in cui $2a^3 > b^2 > 0$.

Esercizio 10 Si descriva un algoritmo per calcolare l'espressione

$$f(a,b,c) = \frac{ac+bc}{a-b}$$

che sia stabile all'indietro e si diano maggiorazioni ai valori assoluti delle perturbazioni $\delta_a, \delta_b, \delta_c$ per cui $\varphi(a,b,c) = f(a(1+\delta_a), b(1+\delta_b), c(1+\delta_c))$, dove $\varphi(a,b,c)$ è il valore calcolato eseguendo l'algoritmo in aritmetica floating point con numeri di macchina a, b, c, $a \neq b$.

Soluzione. Si usa l'espressione $f(a,b,c) = c(a+b)/(a-b)$. Calcolando in aritmetica floating point e denotando con ϵ_i, $i = 1,2,3,4$ rispettivamente gli errori locali generati nell'addizione, sottrazione, divisione e moltiplicazione si ha

$$\varphi(a,b,c) = (c(a+b)/(a-b))(1+\epsilon_1)(1+\epsilon_2)(1+\epsilon_3)(1+\epsilon_4) \doteq f(a,b,c)(1+\epsilon_1+\epsilon_2+\epsilon_3+\epsilon_4).$$

Ponendo quindi $\hat{a} = a$, $\hat{b} = b$, $\hat{c} = c(1+\delta_c)$, con $\delta_c = \epsilon_1+\epsilon_2+\epsilon_3+\epsilon_4$, risulta $\varphi(a,b,c) \doteq f(\hat{a},\hat{b},\hat{c})$. Vale inoltre $|\delta_c| < 4u$. Si osservi inoltre che, per definizione, indipendentemente dall'analisi all'indietro svolta, risulta $\epsilon_{\text{alg}} := \epsilon_1+\epsilon_2+\epsilon_3+\epsilon_4$.

Esercizio 11 Si consideri la successione $\{x_k\}$ definita da

$$x_{k+1} = a_k(x_k^2) + x_{k-1}/b_k, \quad k = 1, 2, \ldots,$$

dove $x_0 = x_1 = 1$ e a_k, b_k sono numeri di macchina. Siano $\tilde{x}_k$ i valori ottenuti applicando la formula in aritmetica floating point con precisione u. Si dimostri che esistono ϵ_k, δ_k, $k = 1, 2, \ldots$, tali che

$$\tilde{x}_{k+1} = \hat{a}_k(\tilde{x}_k^2) + \tilde{x}_{k-1}/\hat{b}_k, \quad k = 1, 2, \ldots,$$

con $\tilde{x}_0 = \tilde{x}_1 = 1$, $\hat{a}_k = a_k(1 + \delta_a^{(k)})$, $\hat{b}_k = b_k(1 + \delta_b^{(k)})$. Si diano maggiorazioni al primo ordine in funzione di u a $|\delta_a^{(k)}|$ e $|\delta_b^{(k)}|$.

Soluzione. Si calcola il passo k-esimo nel modo seguente

$$\begin{aligned} s_1 &= x_k \cdot x_k \\ s_2 &= a_k \cdot s_1 \\ s_3 &= x_{k-1}/b_k \\ x_{k+1} &= s_2 + s_3 \end{aligned}$$

Operando in aritmetica floating point e denotando $\tilde{x}_k$ e $\tilde{x}_{k-1}$ le quantità calcolate in aritmetica floating point nei passi precedenti, si ottiene

$$\begin{aligned} \tilde{s}_1 &= \tilde{x}_k \cdot \tilde{x}_k(1 + \epsilon_1^{(k)}) \\ \tilde{s}_2 &= a_k \cdot \tilde{s}_1(1 + \epsilon_2^{(k)}) \\ \tilde{s}_3 &= (\tilde{x}_{k-1}/b_k)(1 + \epsilon_3^{(k)}) \\ \tilde{x}_{k+1} &= (\tilde{s}_2 + \tilde{s}_3)(1 + \epsilon_4^{(k)}) \end{aligned}$$

dove $\epsilon_i^{(k)}$, $i = 1, 2, 3, 4$ sono gli errori locali generati nelle corrispondenti operazioni aritmetiche. Si ottiene quindi

$$\tilde{x}_{k+1} = \left[a_k\tilde{x}_k^2(1 + \epsilon_1^{(k)})(1 + \epsilon_2^{(k)}) + (\tilde{x}_{k-1}/b_k)(1 + \epsilon_3^{(k)})\right](1 + \epsilon_4^{(k)})$$

Ponendo quindi $\hat{a}_k = a_k(1 + \epsilon_1^{(k)})(1 + \epsilon_2^{(k)})(1 + \epsilon_4^{(k)}) =: a_k(1 + \delta_a^{(k)})$ e $\hat{b}_k = b_k/((1 + \epsilon_3^{(k)})(1 + \epsilon_4^{(k)})) =: b_k(1 + \delta_b^{(k)})$, risulta

$$\tilde{x}_{k+1} = \hat{a}_k\tilde{x}_k^2 + \tilde{x}_{k-1}/\hat{b}_k$$

inoltre $\delta_a^{(k)} \doteq \epsilon_1^{(k)} + \epsilon_2^{(k)} + \epsilon_4^{(k)}$, $\delta_b^{(k)} \doteq -\epsilon_3^{(k)} - \epsilon_4^{(k)}$, per cui $|\delta_a^{(k)}| \dot{\leq} 3u$, $|\delta_b^{(k)}| \dot{\leq} 2u$. □

Esercizio 12 Per calcolare la funzione $f(x_1, ..., x_n) = \sum_{k=1}^{n} \prod_{j=1}^{k} x_j$ si consideri l'algoritmo seguente

$$s_1 = x_1, \quad s_i = (1 + s_{i-1})x_i, \quad i = 2, \ldots, n.$$

a) Si dimostri che l'esecuzione dell'algoritmo in aritmetica floating point genera dei numeri di macchina $\tilde{s}_i$ tali che $\tilde{s}_i = (1 + \tilde{s}_{i-1})\tilde{x}_i$, per $i = 2, \ldots, n$ dove $\tilde{x}_i = x_i(1 + \epsilon_i)$, $|\epsilon_i| \dot{\leq} 2u$, $\tilde{s}_1 = s_1 = x_1$ e u è la precisione dell'aritmetica.

b) Nell'ipotesi che i dati x_i siano compresi tra 0 e 1, si maggiorino i coefficienti di amplificazione di $f(x_1, ..., x_n)$ e si usi il risultato del punto a) per dare una maggiorazione al primo ordine dell'errore algoritmico.

Soluzione. a) Dimostriamo la proprietà per induzione. Se $n = 2$, vale

$$\tilde{s}_2 = ((1 + s_1)(1 + \alpha_1))x_2(1 + \alpha_2)$$

dove α_1 e α_2 sono gli errori locali generati dalle singole operazioni aritmetiche, $|\alpha_1| < u$, $|\alpha_2| < u$. Dunque $\tilde{s}_2 = (1 + \tilde{s}_1)\tilde{x}_2$, dove $\tilde{x}_2 = x_2(1 + \epsilon_2)$ e $|\epsilon_2| \doteq |\alpha_1 + \alpha_2| < 2u$ e $\tilde{s}_1 = s_1 = x_1$.

Supponiamo che sia $\tilde{s}_i = (1 + \tilde{s}_{i-1})\tilde{x}_i$, con $\tilde{x}_i = x_i(1 + \epsilon_i)$ e $|\epsilon_i| \dot{\leq} 2u$, per $i = 2, \ldots, n - 1$. Allora

$$\tilde{s}_n = ((1 + \tilde{s}_{n-1})(1 + \alpha_{1,n}))x_n(1 + \alpha_{2,n})$$

dove $\alpha_{1,n}$ e $\alpha_{2,n}$ sono gli errori locali generati dalle singole operazioni aritmetiche, $|\alpha_{1,n}| < u$, $|\alpha_{2,n}| < u$. Da cui $\tilde{s}_n = (1 + \tilde{s}_{n-1})\tilde{x}_n$, dove $\tilde{x}_n = x_n(1 + \epsilon_n)$ e $|\epsilon_n| \doteq |\alpha_{1,n} + \alpha_{2,n}| < 2u$.

b) Poiché il valore effettivamente calcolato è $f(\tilde{x}_1, \ldots, \tilde{x}_n)$, l'errore algoritmico è dato da $\epsilon_{\text{alg}} \doteq \sum_{i=1}^{n} c_i\epsilon_i$ dove $c_i = \frac{x_i}{f(x_1,\ldots,x_n)}\frac{\partial f}{\partial x_i}$. Dalla definizione di $f(x_1, \ldots, x_n)$ segue che, poiché $x_i > 0$ per $i = 1, \ldots, n$, $0 < \frac{\frac{\partial f}{\partial x_i}}{f(x_1,\ldots,x_n)} < 1$ per ogni i. Dunque, poiché $0 < x_i < 1$ e $|\epsilon_i| \dot{\leq} 2u$, si ottiene $\epsilon_{\text{alg}} \dot{\leq} 2nu$. □

Esercizio 13 Dati un intero $n > 1$ e tre vettori $a = (a_i)$, $c = (c_i) \in \mathbb{R}^n$, $b = (b_i) \in \mathbb{R}^{n-1}$ si consideri il sistema $Tx = c$ dove $T = (t_{i,j})$ è la matrice triangolare inferiore tale che $t_{i,i} = a_i$, per $i = 1, \ldots, n$, $t_{i+1,i} = b_i$, $i = 1, \ldots, n-1$, e $t_{i,j} = 0$ altrove. Descrivere un algoritmo per la risoluzione del sistema che impieghi al più $3n$ operazioni aritmetiche. Fare l'analisi all'indietro dell'errore e scrivere una function nella sintassi di Octave che implementi l'algoritmo.

Esercizio 14 Siano $u, v \in \mathbb{R}^n$ con $u_1 = v_1$ e si definisca la matrice $n \times n$ $A_n = (a_{i,j})$ tale che $a_{i,i+1} = 1$, $i = 1, \ldots, n-1$, $a_{i,i} = u_i$, $a_{i,1} = v_i$, $i = 1, \ldots, n$. Si denoti $d_n = \det A_n$.

a) Scrivere una relazione che lega d_n con d_{n-1} e ricavarne un algoritmo per il calcolo di d_n che impieghi non più di $2n$ operazioni aritmetiche.

b) Scrivere una function con la sintassi di Octave che implementi l'algoritmo del punto a).

c) Dimostrare la stabilità all'indietro dell'algoritmo.

Soluzione. a) Sviluppando il determinante della matrice A_n, con $n \geq 2$, rispetto all'ultima riga, si ottiene che $d_n = u_n d_{n-1} + (-1)^{n+1} v_n$. Dunque d_n può essere calcolato mediante l'algoritmo

$$\begin{aligned} &d_1 = u_1 \\ &d_i = u_i d_{i-1} + (-1)^{i+1} v_i, \quad i = 2, \ldots, n \end{aligned}$$

che impiega $2(n-1)$ operazioni aritmetiche.

c) Sia $\tilde{d}_i$ il valore effettivamente calcolato di d_i. Vale

$$\begin{aligned} &\tilde{d}_1 = d_1 = u_1 \\ &\tilde{d}_i = \big((u_i \tilde{d}_{i-1})(1+\alpha_i) + (-1)^{i+1} v_i\big)(1+\beta_i), \quad i = 2, \ldots, n \end{aligned}$$

dove α_i, β_i, con $|\alpha_i|, |\beta_i| < u$, sono gli errori locali. Dunque

$$\tilde{d}_i = \tilde{u}_i \tilde{d}_{i-1} + (-1)^{i+1} \tilde{v}_i, \quad i = 2, \ldots, n$$

dove $\tilde{u}_i = u_i(1+\alpha_i)(1+\beta_i)$, $\tilde{v}_i = v_i(1+\alpha_i)$. Dunque il valore effettivamente calcolato è il determinante della matrice definita di vettori $\tilde{u}$, $\tilde{v}$, con elementi $\tilde{u}_i$ e $\tilde{v}_i$, rispettivamente. □

Esercizio 15 Sono dati tre vettori $b = (b_i), u = (u_i), v = (v_i) \in \mathbb{R}^n$, con $b_1 = u_1 = v_1$. Si consideri la matrice $A_n = (a_{i,j})$ di dimensione $n \times n$ con $a_{i,i} = b_i$, $i = 1, \ldots, n$, $a_{1,i} = u_i$, $a_{i,1} = v_i$, $i = 2, \ldots, n$. Si ponga $d_n = \det A_n$.

a) Si scriva la relazione che lega d_n e d_{n-1}

b) Si implementi tale relazione in una function nella sintassi di Octave che prende come input i tre vettori b, u, v e dà in output il vettore $d = (d_i) \in \mathbb{R}^n$

c) Si dica se tale formula è stabile all'indietro.

Esercizio 16 Scrivere una function nella sintassi di Octave che data una matrice $n \times n$, reale A con elementi diagonali nulli, calcoli la somma dei determinanti di tutte le sottomatrici principali 2×2 di A. Svolgere un'analisi dell'errore dell'algoritmo implementato dando una limitazione superiore all'errore algoritmico nell'ipotesi che A abbia elementi compresi tra 0 e 1.

Esercizio 17 La funzione $f(x) = \frac{1}{x(1-x)}$ può essere scritta come $f(x) = \frac{1}{x} - \frac{1}{1-x}$. Si analizzi l'errore algoritmico nel calcolo di $f(x)$ con i metodi ottenuti dalle due diverse rappresentazioni. Dire quale dei due metodi è numericamente più stabile.

Esercizio 18 Dato un intero $n > 0$ e assegnati i numeri reali positivi a_i, b_i, c_i per $i = 0, 1, \ldots, n$, si definisca $x_{i+1} = a_i + c_i b_i / x_i$, per $i = 0, 1, \ldots n$, dove $x_0 > 0$ è assegnato. Siano $\tilde{x}_i$ i valori ottenuti calcolando gli x_i con aritmetica *floating point* con precisione u. Dire se esistono perturbazioni $\alpha_i, \beta_i, \gamma_i$ tali che posto $\tilde{a}_i = a_i(1+\alpha_i)$, $\tilde{b}_i = b_i(1+\beta_i)$, $\tilde{c}_i = c_i(1+\gamma_i)$, risulta $\tilde{x}_{i+1} = \tilde{a}_i + \tilde{c}_i \tilde{b}_i / \tilde{x}_i$; in tal caso si diano maggiorazioni a $|\alpha_i|, |\beta_i|, |\gamma_i|$ in funzione di u.
Svolgere una analisi analoga nel caso del calcolo di $x_{i+1} = a_i + c_i b_i / (a_i x_i)$.

Esercizio 19 Sia $n > 0$ intero e sia $s_n = s_n(a_1, \ldots, a_n, b_1, \ldots, b_n)$ tale che

$$s_k = a_k s_{k-1} + b_k / s_{k-1} + b_k, \quad k = 1, \ldots, n, \quad s_0 = 1.$$

Determinare un algoritmo stabile all'indietro per il calcolo di s_n. Denotando $\widetilde{s}_k$ i valori calcolati dall'algoritmo in aritmetica *floating point*, dare maggiorazioni a $|\alpha_k|$ e $|\beta_k|$ per cui $\widetilde{s}_n = s_n(\widetilde{a}_1, \ldots, \widetilde{a}_n, \widetilde{b}_1, \ldots, \widetilde{b}_n)$, dove $\widetilde{a}_k = a_k(1 + \alpha_k)$, $\widetilde{b}_k = b_k(1 + \beta_k)$.

Esercizio 20 Siano $f(x), g(x) : \mathbb{R} \to \mathbb{R}$ funzioni razionali e $\varphi(\xi)$, $\gamma(\eta)$ i risultati forniti da due algoritmi per il loro calcolo applicati in aritmetica *floating point* con precisione u a partire da numeri di macchina ξ, η. Si assuma che in assenza di *overflow* o *underflow*, esistano $\delta_1, \delta_2 \in \mathbb{R}$ tali che $\varphi(\xi) = f(\xi(1 + \delta_1))$, $\gamma(\eta) = g(\eta(1 + \delta_2))$, dove $|\delta_1| \leq u\theta$ e $|\delta_1| \leq u\theta$, $\theta > 0$. Dimostrare che esiste $\epsilon \in \mathbb{R}$ tale che $\gamma(\varphi(\xi)) = g(f(\xi(1 + \epsilon)))$, in assenza di *overflow* o *underflow*, e dare maggiorazioni a $|\epsilon|$ al primo ordine in funzione di u.

Esercizio 21 Descrivere un algoritmo numericamente stabile all'indietro per il calcolo di $f(x, y, z) = x/(yz) + y/(xz)$ e farne l'analisi all'indietro dell'errore. Dire se l'algoritmo trovato può generare errori di cancellazione numerica per $z > 0$.

Soluzione. Vale $f(x, y, z) = (x/y + y/x)/z$ per cui si può usare il seguente algoritmo

$$\begin{aligned} s_1 &= x/y \\ s_2 &= y/x \\ s_3 &= s_1 + s_2 \\ f &= s_3/z \end{aligned}$$

Eseguendo l'algoritmo in aritmetica floating point si ha

$$\begin{aligned} \tilde{s}_1 &= (x/y)(1 + \epsilon_1) \\ \tilde{s}_2 &= (y/x)(1 + \epsilon_2) \\ \tilde{s}_3 &= (\tilde{s}_1 + \tilde{s}_2)(1 + \epsilon_3) \\ \tilde{f} &= (\tilde{s}_3/z)(1 + \epsilon_4) \end{aligned}$$

dove ϵ_i, $i = 1, 2, 3, 4$ sono gli errori locali generati nelle corrispondenti operazioni aritmetiche. Si ottiene allora

$$\tilde{f} = \frac{x}{yz}(1+\epsilon_1)(1+\epsilon_3)(1+\epsilon_4) + \frac{y}{xz}(1+\epsilon_2)(1+\epsilon_3)(1+\epsilon_4) \doteq \frac{x}{yz}(1+\epsilon_1+\epsilon_3+\epsilon_4) + \frac{y}{xz}(1+\epsilon_2+\epsilon_3+\epsilon_4).$$

Cerchiamo ora perturbazioni $\delta_x, \delta_y, \delta_z$ tali che i valori $\hat{x} = x(1 + \delta_x)$, $\hat{y} = y(1 + \delta_y)$, $\hat{z} = z(1 + \delta_z)$, soddisfino la condizione $\tilde{f} = f(\hat{x}, \hat{y}, \hat{z})$. Imponendo questa condizione sulla parte lineare degli errori si ottiene

$$\begin{aligned} \delta_x - \delta_y - \delta_z &= \epsilon_1 + \epsilon_3 + \epsilon_4 \\ \delta_y - \delta_x - \delta_z &= \epsilon_2 + \epsilon_3 + \epsilon_4. \end{aligned}$$

Sommando e sottraendo le due espressioni si ottiene il sistema equivalente

$$\begin{aligned}\delta_z &= -(\epsilon_3 + \epsilon_4 + (\epsilon_1 + \epsilon_2)/2)\\ \delta_x - \delta_y &= \epsilon_1/2 - \epsilon_2/2\end{aligned}$$

che è risolto da $\delta_x = \epsilon_1/2$, $\delta_y = \epsilon_2/2$, $\delta_z = -(\epsilon_3 + \epsilon_4 + (\epsilon_1 + \epsilon_2)/2)$, per cui $|\delta_x| < u/2$, $|\delta_y| < u/2$, $|\delta_z| < 3u$. □

Esercizio 22 Descrivere un algoritmo numericamente stabile all'indietro per il calcolo di $f(x, y) = x^2/y + x$ e farne l'analisi all'indietro dell'errore. Utilizzare tale analisi per determinare limitazioni superiori al valore assoluto dell'errore algoritmico nel caso $xy > 0$.

Soluzione. Usando l'espressione $f(x, y) = x(1 + x/y)$ si può calcolare f col seguente algoritmo

$$\begin{aligned}s_1 &= x/y,\\ s_2 &= s_1 + 1,\\ f &= x \cdot s_2.\end{aligned}$$

Operando in aritmetica floating point si ottiene

$$\begin{aligned}\tilde{s}_1 &= (x/y)(1 + \epsilon_1),\\ \tilde{s}_2 &= (\tilde{s}_1 + 1)(1 + \epsilon_2),\\ \tilde{f} &= x \cdot \tilde{s}_2(1 + \epsilon_3),\end{aligned}$$

dove $\epsilon_1, \epsilon_2, \epsilon_3$ sono gli errori locali generati dalle tre operazioni aritmetiche. Vale quindi

$$\tilde{f} = x(1 + (x/y)(1 + \epsilon_1))(1 + \epsilon_3)(1 + \epsilon_2)$$

Per cui, posto $\hat{x} = x(1 + \epsilon_3)(1 + \epsilon_2)$ $\hat{y} = y(1 + \epsilon_3)(1 + \epsilon_2)/(1 + \epsilon_1)$ vale

$$\tilde{f} = f(\hat{x}, \hat{y}).$$

Inoltre vale $\hat{x} = x(1 + \delta_1)$, $\hat{y} = y(1 + \delta_2)$, $\delta_1 \doteq \epsilon_2 + \epsilon_3$, $\delta_2 \doteq \epsilon_3 - \epsilon_1 - \epsilon_2$, per cui $|\delta_1| \stackrel{\cdot}{\leq} 2u$, $|\delta_2| \stackrel{\cdot}{\leq} 3u$.

Per quanto riguarda l'errore algoritmico vale

$$\epsilon_{\text{alg}} \doteq C_x\delta_1 + C_y\delta_2$$

dove $C_x = (2x^2/y + x)/(x^2/y + x)$, $C_y = -(x^2/y)/(x^2/y + x)$ sono i coefficienti di amplificazione di $f(x, y)$. Risulta allora

$$|\epsilon_{\text{alg}}| \stackrel{\cdot}{\leq} u(4|x^2/y| + 2|x| + 3|x^2/y|)/|x^2/y + x|$$

Se $xy > 0$ si ottiene

$$|\epsilon_{\text{alg}}| \stackrel{\cdot}{\leq} u|7x^2/y + 2x|/|x^2/y + x| = u(7 - 5/(x/y + 1)) < 7u.$$

Un'altra possibilità è usare l'algoritmo

$$\begin{aligned} s_1 &= x \cdot x, \\ s_2 &= s_1/y, \\ f &= s_2 + x. \end{aligned}$$

Procedendo in modo analogo, il valore effettivamente calcolato in aritmetica floating point è dato da

$$\tilde{f} = \left[(x^2/y)(1+\epsilon_1)(1+\epsilon_2) + x\right](1+\epsilon_3)$$

Ponendo allora $\hat{x} = x(1+\epsilon_3)$ e $\hat{y} = y(1+\epsilon_3)/((1+\epsilon_1)(1+\epsilon_2))$ si ha $\tilde{f} = f(\hat{x}, \hat{y})$. Vale quindi $\hat{x} = x(1+\delta_1)$, $\hat{y} = y(1+\delta_2)$, $|\delta_1| < u$, $|\delta_2| \stackrel{\cdot}{\leq} 3u$. La limitazione all'errore algoritmico ottenuta in questo modo è quindi migliore e, se $xy > 0$ vale

$$|\epsilon_{\text{alg}}| \stackrel{\cdot}{\leq} u(2|x^2/y| + |x| + 3|x^2/y|)/|x^2/y + x| \leq u(1 + 4|x^2/y|/|x^2/y + x|) < 5u$$

□

Esercizio 23 Descrivere un algoritmo per il calcolo di

$$c + \sum_{i=1,n} a_i/(b_i - x)$$

numericamente stabile all'indietro e farne l'analisi all'indietro dell'errore.

Esercizio 24 Siano a, b, c tre numeri di macchina. Per il calcolo di $ab - bc$ si considerino le seguenti espressioni

$$b(a-c), \quad ab - bc, \quad a(b+c) - c(a+b).$$

Dare limitazioni superiori al valore assoluto dell'errore algoritmico nei tre casi e confrontare.

Esercizio 25 Si considerino le funzioni $f_k = f_k(x_1, \ldots, x_k, y_1, \ldots, y_k)$, $g_k = g_k(x_1, \ldots, x_k, y_1, \ldots, y_k)$, definite da

$$\begin{aligned} f_{k+1} &= x_{k+1}\, f_k\, g_k, \\ g_{k+1} &= (f_k^2 - g_k^2)/y_{k+1} \end{aligned} \qquad k = 0, 1, \ldots, n-1, \tag{7}$$

dove $f_0 = g_0 = 1$. Dare un algoritmo per il calcolo della coppia (f_n, g_n) numericamente stabile all'indietro e farne l'analisi all'indietro dell'errore.

Soluzione. È sufficiente riscrivere la seconda delle due espressioni come $g_{k+1} = (f_k^2 - g_k^2)/y_{k+1} = (f_k - g_k)(f_k + g_k)/y_{k+1}$. Il valore calcolato in aritmetica floating point è

$$\tilde{g}_{k+1} = [(f_k - g_k)(f_k + g_k)/y_{k+1}]\,(1+\epsilon_k)(1+\mu_k)(1+\eta_k),$$

dove $\epsilon_k, \mu_k, \eta_k, \nu_k$ sono gli errori locali generati rispettivamente dalla sottrazione, addizione moltiplicazione e divisione. Si ha quindi

$$\tilde{g}_{k+1} = (f_k - g_k)(f_k + g_k)/\hat{y}_{k+1}$$

con $\hat{y}_{k+1} = y_{k+1}/((1+\epsilon_k)(1+\mu_k)(1+\eta_k)(1+\nu_k)) = y_k(1+\delta_k)$, $\delta_k \doteq -\epsilon_k - \mu_k - \eta_k - \nu_k$ per cui $|\delta_k| \dot{\leq} 4u$. □

Esercizio 26 Dare un algoritmo stabile all'indietro per il calcolo di $f(a,b,c,x) = -ax^{-1} + b + ax + cx^2$ e farne l'analisi all'indietro dell'errore.

Soluzione. Vale $f(a,b,c,x) = a(x-1)(x+1)/x + b + cx^2$ da cui si ricava l'algoritmo sintetizzato dalla formula

$$f(a,b,c,x) = ((cx^2 + b) + a(x-1)(x+1)/x)$$

che applicato in aritmetica floating point produce

$$\begin{aligned}\tilde{f} =& cx^2(1+\epsilon_1)(1+\epsilon_2)(1+\epsilon_3)(1+\epsilon_4)\\ &+b(1+\epsilon_3)(1+\epsilon_4)\\ &+a(x-1)(x+1)/x(1+\epsilon_4)(1+\theta_1)(1+\theta_2)(1+\theta_3)(1+\theta_4)(1+\theta_5),\end{aligned}$$

dove $\epsilon_1, \ldots, \epsilon_4$ sono gli errori locali commessi nell'esecuzione delle tre operazioni aritmetiche nel calcolo di $cx^2 + b$ e nell'operazione di addizione della quantità così ottenuta con l'addendo successivo. Mentre $\theta_1, \ldots, \theta_5$ sono gli errori locali commessi nell'esecuzione delle 5 operazioni aritmetiche nel calcolo del secondo addendo $a(x-1)(x+1)/x$. L'analisi all'indietro si completa ponendo

$$\begin{aligned}\hat{c} &= c(1+\epsilon_1)(1+\epsilon_2)(1+\epsilon_3)(1+\epsilon_4) =: c(1+\delta_c),\\ \hat{b} &= b(1+\epsilon_3)(1+\epsilon_4) =: b(1+\delta_b),\\ \hat{a} &= a(1+\epsilon_4)(1+\theta_1)(1+\theta_2)(1+\theta_3)(1+\theta_4)(1+\theta_5)(1+\epsilon_4) =: a(1+\delta_a)\end{aligned}$$

Vale quindi $|\delta_a| \dot{\leq} 7u$, $|\delta_b| \dot{\leq} 2u$, $|\delta_c| \dot{\leq} 4u$. □

Esercizio 27 Si considerino le successioni $\{x_k\}$ e $\{y_k\}$ definite da

$$x_{k+1} = (x_k + y_k)/2, \qquad y_{k+1} = \sqrt{x_k y_k}$$

a partire da $x_0, y_0 > 0$. Si dia una maggiorazione a $\theta_k = \max(|\xi_k|, |\eta_k|)$ dove ξ_k e η_k sono gli errori algoritmici nel calcolo rispettivamente di x_k e y_k in aritmetica *floating point* con precisione u, dove la radice quadrata viene calcolata da una funzione "di macchina" SQRT tale che $\mathrm{SQRT}(w) = \sqrt{w}(1+\epsilon)$, $|\epsilon| < u$. Se $u = 2^{-52}$ qual è il massimo valore di n per cui ci sono almeno 32 bit corretti nei valori effettivamente calcolati di x_n e y_n?

Esercizio 28 Descrivere un algoritmo stabile all'indietro per il calcolo di $f(x,y,z) = (xy+yz+zx)/(xyz)$ e farne l'analisi all'indietro dell'errore.

Soluzione Vale $f(x,y,z) = 1/z + 1/x + 1/y$ per cui un algoritmo di calcolo è dato da:

$$\begin{aligned} s_1 &= 1/z \\ s_2 &= 1/x \\ s_3 &= 1/y \\ s_4 &= s_1 + s_2 \\ f &= s_4 + s_3 \end{aligned}$$

Una sua esecuzione in aritmetica floating point genera quantità date da

$$\begin{aligned} \tilde{s}_1 &= (1/z)(1+\epsilon_1) \\ \tilde{s}_2 &= (1/x)(1+\epsilon_2) \\ \tilde{s}_3 &= (1/y)(1+\epsilon_3) \\ \tilde{s}_4 &= (\tilde{s}_1 + \tilde{s}_2)(1+\epsilon_4) \\ \tilde{f} &= (\tilde{s}_4 + \tilde{s}_3)(1+\epsilon_5) \end{aligned}$$

dove ϵ_i, $i = 1, \ldots, 5$ sono gli errori locali generati rispettivamente nelle corrispondenti operazioni aritmetiche. Vale quindi

$$\tilde{f} = (1+\epsilon_5)(1+\epsilon_4)(1+\epsilon_1)\frac{1}{z} + (1+\epsilon_5)(1+\epsilon_4)(1+\epsilon_2)\frac{1}{x} + (1+\epsilon_5)(1+\epsilon_3)\frac{1}{y}$$

da cui $\tilde{f} = f(\hat{x}, \hat{y}, \hat{z})$ con $\hat{x} = x/((1+\epsilon_5)(1+\epsilon_4)(1+\epsilon_2)) =: x(1+\delta_1)$, $\hat{y} = y/((1+\epsilon_5)(1+\epsilon_3)) =: y(1+\delta_2)$ $\hat{z} = z/((1+\epsilon_5)(1+\epsilon_4)(1+\epsilon_1)) =: z(1+\delta_3)$, dove $\delta_1 \doteq -\epsilon_5 - \epsilon_4 - \epsilon_2$, $\delta_2 \doteq -\epsilon_5 - \epsilon_3$, $\delta_3 \doteq -\epsilon_5 - \epsilon_4 - \epsilon_1$. Per cui $|\delta_1| \dot{\leq} 3u$, $|\delta_2| \dot{\leq} 2u$, $|\delta_3| \dot{\leq} 3u$. □

Esercizio 29 Dare limitazioni superiori al valore assoluto degli errori algoritmici commessi nel calcolo di $f(x) = x^2 + x + 1$ mediante le due espressioni

$$f(x) = x(x+1) + 1 = (x+1)^2 - x$$

in aritmetica *floating point* con precisione di macchina u. Confrontare le due limitazioni così ottenute per $x > 0$.

Soluzione. Detto $\tilde{f}$ il valore effettivamente calcolato nel primo algoritmo si ha

$$\tilde{f} = (x(x+1)(1+\nu)(1+\eta) + 1)(1+\mu)$$

dove ν, η, μ sono gli errori locali generati rispettivamente dalla prima addizione, dalla moltiplicazione e dalla seconda addizione. Per cui l'errore algoritmico diventa

$$\epsilon_{alg_1} \doteq \frac{x(x+1)(\eta+\nu+\mu)+\mu}{x^2+x+1}$$

da cui $|\epsilon_{alg_1|} \dot{\leq} u\frac{3|x(x+1)|+1}{|x^2+x+1|}$

Detto $\hat{f}$ il valore effettivamente calcolato nel secondo algoritmo si ha

$$\tilde{f} = ((x+1)^2(1+\mu)^2(1+\nu)-x)(1+\eta)$$

dove μ è l'errore locale della prima addizzione, ν è l'errore locale dell'elevamento a quadrato, mentre η è l'errore locale della sottrazione. Per cui l'errore algoritmico diventa

$$\epsilon_{alg_2} \doteq \frac{(x+1)^2(\eta+\nu+2\mu)+\eta x}{x^2+x+1}$$

da cui $|\epsilon_{alg_2}| \dot{\leq} u\frac{4(x+1)^2+|x|}{|x^2+x+1|}$.

Il primo algoritmo è più conveniente del secondo se $3|x(x+1)|+1 < 4(x+1)^2+|x|$, Cioè se $x < -3$ oppure $x > -3/7$.

L'analisi dell'errore algoritmico poteva essere svolta equivalentementw costruendo il grafo di calcolo associato ai due algoritmi. □

Esercizio 30 Si considerino le funzioni $f_k = f_k(x_1, \ldots, x_k, y_1, \ldots, y_k)$, $g_k = g_k(x_1, \ldots, x_k, y_1, \ldots, y_k)$, definite da

$$\begin{aligned} f_{k+1} &= f_k\, g_k/x_{k+1}, \\ g_{k+1} &= y_{k+1}\,(f_k^2+g_k^2-2g_k\, f_k), \end{aligned} \qquad k = 0, 1, \ldots, n-1, \tag{8}$$

dove $f_0 = g_0 = 1$. Dare un algoritmo numericamente stabile all'indietro per il calcolo della coppia (f_n, g_n) e farne l'analisi all'indietro dell'errore.

Soluzione. I valori effettivamente calcolati mediante le espressioni $f_{k+1} = (f_k g_k)/x_{k+1}$ e $g_{k+1} = y_{k+1}(f_k - g_k)^2$, generano numeri di macchina $\tilde{f}_k$ e $\tilde{g}_k$ tali che

$$\begin{aligned} \tilde{f}_{k+1} &= \tilde{f}_k\tilde{g}_k(1+\alpha_k)(1+\beta_k)/x_{k+1} \\ \tilde{g}_{k+1} &= y_{k+1}(\tilde{f}_k-\tilde{g}_k)^2(1+\gamma_k)(1+\eta_k)(1+\nu_k) \end{aligned}$$

dove α_k e β_k sono gli errori locali generati dalla moltiplicazione e dalla divisione nel calcolo di $\tilde{f}_{k+1}$, mentre γ_k, η_k, ν_k sono gli errori locali generati rispettivamente dalla addizione, l'elevamento a quadrato e dalla moltiplicazione per y_{k+1}. Per cui i valori assoluti di questi errori locali sono maggiorati dalla precisione di macchina u. Quindi, ponendo $\hat{x}_{k+1} = x_{k+1}/((1+\alpha_k)(1+\beta_k))$ e $\hat{y}_{k+1} = y_{k+1}(1+\gamma_k)(1+\eta_k)(1+\nu_k)$ si ottiene la relazione

$$\begin{aligned} \tilde{f}_{k+1} &= \tilde{f}_k\tilde{g}_k/\hat{x}_{k+1} \\ \tilde{g}_{k+1} &= \hat{y}_{k+1}(\tilde{f}_k-\tilde{g}_k)^2 \end{aligned}$$

che dimostra la stabilità all'indietro dell'algoritmo. Inoltre vale $\hat{x}_{k+1} \doteq x_{k+1}(1-\alpha_k-\beta_k) =: x_{k+1}(1+\delta_k)$, $|\delta_k| < 2u$, $\hat{y}_{k+1} \doteq y_{k+1}(1+\gamma_k)(1+\eta_k)(1+\nu_k) =: y_{k+1}(1+\delta'_k)$, $|\delta'_k| < 3u$. □

Esercizio 31 Dato un intero pari $n = 2m > 0$ e numeri reali a_i, f_i, $i = 1, \ldots, n$, b_i, $i = 1, \ldots, n-1$, si consideri il sistema lineare $Hx = f$, dove $H = (h_{i,j})$ è la matrice $n \times n$ tale che gli elementi non nulli sono tutti e soli gli $h_{i,i} = a_i$, per $i = 1, \ldots, n$, gli $h_{2i-1,2i} = b_{2i-1}$, per $i = 1, \ldots, m$ e gli $h_{2i+1,2i} = b_{2i}$, per $i = 1, m-1$.

a) Si descriva un algoritmo per risolvere il sistema $Hx = f$ che impieghi non più di $3n$ operazioni aritmetiche, e se ne faccia l'analisi all'indietro dell'errore.

Esercizio 32 Dato un intero $n > 2$ e i vettori $u = (u_i), f = (f_i) \in \mathbb{R}^n$, $v = (v_i) \in \mathbb{R}^{n-1}$, sia $A = (a_{i,j})$ la matrice bidiagonale inferiore con elementi diagonali $a_{i,i} = u_i$, $i = 1, \ldots, n$ ed elementi sottodiagonali $a_{i+1,i} = v_i$, $i = 1, \ldots, n-1$.

Si descriva un algoritmo per risolvere il sistema $Ax = f$ che impieghi non più di $3n$ operazioni aritmetiche e si svolga una analisi all'indietro dell'errore.

Esercizio 33 Dato un intero $n > 2$ e i vettori $u = (u_i), f = (f_i) \in \mathbb{R}^n$, $v = (v_i) \in \mathbb{R}^{n-1}$, sia $A = (a_{i,j})$ la matrice definita da $a_{i,i} = u_i$, $i = 1, \ldots, n$, $a_{i,1} = v_{i-1}$, $i = 2, \ldots, n$, $a_{i,j} = 0$ altrove. Descrivere un algoritmo per risolvere il sistema $Ax = f$ che impieghi non più di $3n$ operazioni aritmetiche, studiarne la stabilità all'indietro.

Esercizio 34 Si consideri la funzione $f(m,n) = \sum_{i=m}^{n} \frac{(-1)^i}{i}$ se $m \leq n$, mentre $f(m,n) = 0$ se $m > n$. Descrivere un algoritmo per il calcolo di $f(m,n)$, per $m, n > 0$, che abbia un errore algoritmico limitato superiormente da $\gamma(n-m)u$ dove γ è una costante positiva e u è la precisione di macchina.

Esercizio 35 Dati un intero $n > 1$ e tre vettori $a = (a_i)$, $c = (c_i) \in \mathbb{R}^n$, $b = (b_i) \in \mathbb{R}^{n-1}$ si consideri il sistema $Tx = c$ dove $T = (t_{i,j})$ è la matrice triangolare inferiore tale che $t_{i,i} = a_i$, per $i = 1, \ldots, n$, $t_{i+1,i} = b_i$, $i = 1, \ldots, n-1$, e $t_{i,j} = 0$ altrove. Descrivere un algoritmo per la risoluzione del sistema che impieghi al più $3n$ operazioni aritmetiche. Fare l'analisi all'indietro dell'errore.

Esercizio 36 Siano $u, v \in \mathbb{R}^n$ con $u_1 = v_1$ e si definisca la matrice $n \times n$ $A_n = (a_{i,j})$ tale che $a_{i,i+1} = 1$, $i = 1, \ldots, n-1$, $a_{i,i} = u_i$, $a_{i,1} = v_i$, $i = 1, \ldots, n$. Si denoti $d_n = \det A_n$.

a) Scrivere una relazione che lega d_n con d_{n-1} e ricavarne un algoritmo per il calcolo di d_n che impieghi non più di $2n$ operazioni aritmetiche.

b) Scrivere una function con la sintassi di Octave che implementi l'algoritmo del punto a).

c) Dimostrare la stabilità all'indietro dell'algoritmo.

Esercizio 37 Sono dati tre vettori $b = (b_i), u = (u_i), v = (v_i) \in \mathbb{R}^n$, con $b_1 = u_1 = v_1$. Si consideri la matrice $A_n = (a_{i,j})$ di dimensione $n \times n$ con

$a_{i,i} = b_i$, $i = 1, \ldots, n$, $a_{1,i} = u_i$, $a_{i,1} = v_i$, $i = 2, \ldots, n$. Si ponga $d_n = \det A_n$.
a) Si scriva la relazione che lega d_n e d_{n-1}
b) Si implementi tale relazione in una function nella sintassi di Octave che prende come input i tre vettori b, u, v e dà in output il vettore $d = (d_i) \in \mathbb{R}^n$.
c) Si dica se tale formula è stabile all'indietro.

Esercizio 38 Siano $a = (a_i), x = (x_i), y = (y_i) \in \mathbb{R}^n$, $b = (b_i), c = (c_i) \in \mathbb{R}^{n-1}$, A la matrice tridiagonale $n \times n$ con elementi diagonali a_i, per $i = 1, \ldots, n$, sottodiagonali e sopradiagonali rispettivamente b_i, c_i, $i = 1, \ldots, n-1$, tali che $y = Ax$.

a) Scrivere le formule che legano le componenti di y a quelle di x.

b) Dimostrare che il calcolo di y dati x, a, b, c mediante queste formule è numericamente stabile all'indietro.

c) Scrivere una function nella sintassi di octave che presi in input i vettori x, a, b, c fornisce in output il vettore y.

Esercizio 39 Dati i vettori $d = (d_i), u = (u_i), v = (v_i) \in \mathbb{R}^n$ si definisca la matrice $n \times n$ diagonale D con elementi diagonali $d_1, \ldots, d_n$ e si ponga $A = D + uu^T$. Si descriva un algoritmo per il calcolo di $v^T Av$ che impieghi al più $5n$ operazioni aritmetiche. Farne l'analisi all'indietro dell'errore e dare una *function* nella sintassi di Octave che lo implementi.

Riferimenti bibliografici

[1] R. Bevilacqua, D.A. Bini, M. Capovani, O. Menchi. *Metodi Numerici*. Zanichelli, Bologna 1992

[2] N. J. Higham, *Accuracy and Stability of Numerical Algorithms*, SIAM, Philadelphia 2002.