Chapter 2

Numeri e funzioni al calcolatore

In questo capitolo esaminiamo alcuni argomenti relativi al modo in cui un calcolatore elabora dati numerici per mezzo di operazioni aritmetiche oppure attraverso il calcolo di funzioni. Ci occuperemo anche, per alcuni aspetti elementari, dell'interpolazione lineare, trattando argomenti che verranno poi in parte ripresi nei Capitoli 5 e 8.

2.1 Rappresentazione in virgola mobile dei numeri reali

Sia x un numero reale positivo. Come conseguenza della proprietà archimedea dei numeri reali esiste un intero relativo b tale che $10^b > x$. Quindi x può essere scritto come

$$x = a \cdot 10^b \ , \tag{2.1}$$

dove $a = x/10^b$ è un numero reale positivo minore di 1. La rappresentazione (2.22) si chiama *rappresentazione in virgola mobile (floating point).*

Il numero x è dunque rappresentato dalla coppia $(a, b) \in [0, 1) \times \mathbb{Z}$; il numero a si chiama mantissa di x mentre b è l'esponente di x. È evidente che questa rappresentazione di x non è unica (ad esempio, $9 = 0.9 \cdot 10^1 = 0.09 \cdot 10^2$) e che per rappresentare il numero $-x$ sarà sufficiente cambiare di segno la mantissa.

Richiamiamo ora brevemente la nozione di rappresentazione in base q (q intero maggiore di 1) di un numero reale $a \in [0, 1)$. Dati a e q, definiamo i

numeri interi positivi a_1, a_2, a_3, ..., ponendo

$$\begin{aligned} a_1 &= \max\{k \in N : k/q \leq a\} \\ a_2 &= \max\{k \in N : a_1/q + k/q^2 \leq a\} \\ &\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_n &= \max\{k \in N : a_1/q + a_2/q^2 + ... + a_{n-1}/q^{n-1} + k/q^n \leq a\} \\ &\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \end{aligned} \tag{2.2}$$

Si può dimostrare facilmente che

$$\begin{aligned} 0 &\leq a_i \leq q-1 \ , \text{ per ogni } i = 1, 2, ... \\ 0 &\leq a - \sum_{i=1}^{n} a_i/q^i < q^{-n} \ , \text{ per ogni } n \in N \ . \end{aligned} \tag{2.3}$$

Dunque, ogni numero reale $a \in [0, 1)$ può essere rappresentato, con la precisione voluta, dalla sequenza $\{a_1, a_2, ...\}$ generata dall'algoritmo descritto in (2.23). I numeri interi ai costituiscono la rappresentazione in base q di a e si usa la notazione

$$a = 0.a_1a_2a_3... \tag{2.4}$$

Le rappresentazioni più usate sono quelle in base 2,10 e 16 (binaria, decimale, esadecimale). Notiamo esplicitamente che per la rappresentazione in base 2 occorrono solo le cifre 0 e 1.

2.2 Numeri macchina e aritmetica in virgola mobile

I calcolatori digitali dispongono solo di un numero finito e fisso P (lunghezza della parola) di cifre (decimali,binarie,...) per la rappresentazione interna dei numeri.

La lunghezza della parola varia da macchina a macchina e può essere utilizzata in vari modi per memorizzare un numero. Se si vuole memorizzare un numero in una sola parola *(semplice precisione)* sono dunque disponibili un numero M di posizioni di memoria per la mantissa ed un numero E di posizioni per l'esponente, con

$$P = M + E \ .$$

È eventualmente possibile, anche nei personal computers, usare due parole per memorizzare un singolo numero*(doppia precisione)*.

L'insieme dei numeri rappresentati esattamente da un calcolatore è pertanto un sottoinsieme finito A (numeri macchina) dell'insieme dei numeri reali. Più precisamente, A è costituito dai numeri reali x la cui rappresentazione in virgola mobile richiede al più M cifre (decimali, binarie,...) per la mantissa ed al più E cifre per l'esponente.

Esempio. Se $M = 3$ ed $E = 1$, il numero $x = 0.353 \cdot 10^{23}$ non appartiene ad A (occorrono infatti 2 cifre decimali per rappresentare l'esponente). $\triangle$

Per rappresentare i numeri x che non appartengono ad A il calcolatore esegue dunque automaticamente un'operazione di arrotondamento, sostituendo ad ogni $x \in R - A$ il numero macchina più vicino a x, avente lo stesso esponente di x. Denotando con $rd(x)$ tale numero si ha allora:

$$rd(x) \in A, |x - rd(x)| \leq |x - y|, \quad \text{per ogni } y \in A \ .$$

Per trovare $rd(x)$ il calcolatore opera nella maniera seguente: viene scelta dapprima una rappresentazione in virgola mobile di $x > 0$ la cui mantissa a verifichi

$$10^{-1} \leq a < 1 \ ,$$

(ciò si ottiene se l'esponente b in (2.22) è il minimo intero tale che $x < 10^b$). Allora a avrà una rappresentazione decimale del tipo:

$$a = 0.a_1a_2a_3...a_{M-1}a_Ma_{M+1}...$$

con $a_1 \neq 0$, $0 \leq a_i \leq 9$.

Poiché abbiamo supposto di avere a disposizione M cifre per la mantissa ed E cifre per l'esponente, la mantissa di $rd(x)$ verrà definita come:

$$a' = \begin{cases} 0.a_1a_2...a_{M-1}a_M & \text{se} \quad 0 \leq a_{M+1} \leq 4 \\ 10^{-M} + 0.a_1a_2...a_{M-1}a_M & \text{se} \quad 5 \leq a_{M+1} \leq 9 \ . \end{cases}$$

In altri termini, si tronca la rappresentazione di a alla M–esima cifra decimale, aumentandola di una unità nel caso che $5 \leq a_{M+1}$. Di conseguenza,

$$rd(x) = a' \cdot 10^b \ .$$

Esempio. Per $x = 1/8$ si ha $a = 0.125$ e quindi, se $M = 2$, $a' = 0.13$. $\triangle$

L'*errore relativo*

$$|(rd(x) - x)/x|$$

commesso in questa operazione di arrotondamento è valutabile tramite la diseguaglianza:

$$|(rd(x) - x)/x| \leq 5 \cdot 10^{-(M+1)}/|a| \leq 5 \cdot 10^{-M} \ . \tag{2.5}$$

Il numero $5 \cdot 10^{-M}$ si chiama *precisione di macchina*. Osserviamo che l'operazione di arrotondamento non modifica l'esponente di un numero in virgola mobile. Esistono quindi numeri x per cui anche $rd(x) \notin A$: sono numeri il cui esponente è costituito da più di E cifre. È questo il caso di numeri x con $|x|$ molto piccolo oppure molto grande.

A questi casi, che si possono ovviamente verificare anche quando x non è un dato ma bensı il risultato di qualche operazione effettuata nel corso dell'esecuzione di un programma, ci si riferisce, rispettivamente, come underflow e overflow.

Le difficoltà causate dai fenomeni di underflow e overflow nel corso dell'esecuzione di un programma di calcolo possono talvolta essere superate con opportuni cambiamenti di scala nei dati.

L'insieme dei numeri macchina non è chiuso rispetto alle operazioni aritmetiche elementari:

$$x + y, \ x - y, x \cdot y, \ x/y \ .$$

In altre parole, esistono coppie di numeri macchina la cui somma (oppure, sottrazione, moltiplicazione, divisione) non è un numero macchina. Per convincersi di ciò basta pensare che A ha un minimo ed un massimo.

Esempio. Se per un dato calcolatore $M = 3$ ed $E = 1$, il minimo ed il massimo di A sono, rispettivamente, $-0.999 \cdot 10^9$ e $0.999 \cdot 10^9$. Dunque la somma dei due numeri macchina $0.998 \cdot 10^9$ e $0.1 \cdot 10^9$ non appartiene a A, essendo maggiore di $0.999 \cdot 10^9$. △

Il calcolatore esegue pertanto, anzichè le operazioni esatte, le cosiddette operazioni in virgola mobile, definite per ogni x, $y \in A$ dalle formule:

$$\begin{aligned} x +* y &= rd(x + y) \\ x -* y &= rd(x - y) \\ x \cdot * y &= rd(x \cdot y) \\ x / * \, y &= rd(x/y) \ . \end{aligned}$$

Notiamo esplicitamente che tali operazioni non soddisfano le abituali proprietà aritmetiche dei numeri reali.

Esempio. Se $M = 8$ e $E = 1$, scegliendo i numeri x, y, z come segue

$$\begin{aligned} x &= 0.23371258 \cdot 10^{-4} \\ y &= 0.33678429 \cdot 10^{2} \\ z &= -0.33677811 \cdot 10^{2} \end{aligned}$$

si ha

$$x + *(y + *z) = 0.64137126 \cdot 10^{-3} \ ,$$

mentre

$$(x + *y) + *z = 0.641 \cdot 10^{-3} \ .$$

Ancora,

$$x + *y = x \text{ se } |y| < (5 \cdot 10^{-M}|x|)/9, x, y \in A$$

(in relazione a tali fenomeni vedi anche l'esercizio 4.). △

Un programma di calcolo comporta l'esecuzione di una sequenza di operazioni aritmetiche elementari ed il calcolo di funzioni. I risultati parziali vengono utilizzati come dati iniziali per le operazioni successive. È quindi importante, per valutare la bontà del risultato finale, sapere come gli errori sui dati si propagano nel corso dell'esecuzione di tali operazioni. Una analisi accurata del fenomeno di propagazione dell'errore negli algoritmi va oltre gli scopi di questo libro: ci limitiamo qui a fornire i risultati riguardanti *la propagazione dell'errore nelle operazioni aritmetiche elementari.*

Siano x e y diversi da 0, $\widehat{x}$ e $\widehat{y}$ dei valori approssimati di x e y. Indicando con ε_x l'errore relativo di $\widehat{x}$, definito da

$$\varepsilon_x = (\widehat{x} - x)/x \ ,$$

si può dimostrare che

$$\begin{aligned} \varepsilon_{xy} &= \varepsilon_x + \varepsilon_y \\ \varepsilon_{x/y} &= \varepsilon_x - \varepsilon_y, \text{ con } y \neq 0 \\ \varepsilon_{x\pm y} &= x\frac{\varepsilon_x}{(x \pm y)} \pm y\frac{\varepsilon_y}{(x \pm y)} \ , \quad (x \pm y \neq 0) \ . \end{aligned} \tag{2.6}$$

Le formule (2.28) mostrano che l'operazione più pericolosa dal punto di vista della propagazione dell'errore è la sottrazione fra due numeri x e y dello stesso segno e molto vicini, cioè con $x \cong y$.

2.3 Algoritmo di Ruffini–Horner

I linguaggi comunemente usati sui personal computers contengono delle routines per il calcolo delle funzioni elementari più usate (ad esempio, $\sin x$, $\cos x$, $\exp x$, $\log x$) all'interno delle librerie matematiche

Tali routines si basano su diversi algoritmi che dipendono dalla funzione in oggetto e variano da linguaggio a linguaggio (o addirittura da versione a versione). Anche se una discussione approfondita dell'argomento esula dagli scopi di questo libro, è tuttavia opportuno fare una breve riflessione sul problema del calcolo accurato del valore di una funzione in un punto.

Consideriamo il problema del calcolo del valore di un polinomio p in un assegnato valore di x. Cominciamo con l'osservare che se p è di grado 2, cioè:

$$p(x) = a_2x^2 + a_1x + a_0 \ ,$$

ci sono due modi diversi di calcolare $p(x)$. Descriviamoli:

(A1) calcolare:	(A2) calcolare:
$y_0 = x^2$	$z_0 = a_2x$
$y_1 = a_2y_0$	$z_1 = z_0 + a_1$
$y_2 = a_1x$	$z_2 = z_1x$
$y_3 = p(x) = a_0 + y_2 + y_1$	$z_3 = p(x) = a_0 + z_2$.

Entrambi gli algoritmi forniscono in quattro passi la risposta desiderata, ma (A1) richiede 3 moltiplicazioni e 2 addizioni, mentre per (A2) occorrono solamente 2 moltiplicazioni e 2 addizioni. Dunque, l'algoritmo (A2) è più efficiente dal punto di vista computazionale e porta ad un risparmio anche nel caso di un polinomio di grado 2. Esso è basato sulla diversa (ma equivalente) scrittura di $p(x)$ come:

$$p(x) = (a_2x + a_1)x + a_0 \ .$$

Questa osservazione si estende a polinomi di grado n qualsiasi, dove il guadagno in termini di operazioni elementari sarà maggiore. La formula nel caso generale è:

$$p_n(x) = a_nx^n + a_{n-1}x^{n-1} + ... + a_0 = (...((a_nx + a_{n-1})x + a_{n-2})x + ... + a_0) \ .$$

Su questa seconda espressione del polinomio p_n è basato lo schema di Ruffini–Horner per il calcolo del valore $p_n(x)$ mediante n moltiplicazioni e altrettante addizioni, quindi in totale $2n$ operazioni elementari.

L'algoritmo è il seguente:

$$\begin{aligned} v_n &= a_n \\ v_k &= v_{k+1}x + a_k \ , \quad k = n-1, n-2, ..., 1, 0 \ . \end{aligned}$$

Il numero v_0 cosı ottenuto è il valore $p_n(x)$ cercato (vedi anche esercizi 11 e 12).

2.4 Interpolazione polinomiale di Lagrange

Supponiamo di conoscere i valori di una funzione f definita in un intervallo $[a, b]$ solo in un numero finito di punti $x_0, ..., x_n$ appartenenti all'intervallo $[a, b]$. I punti x_i, $i = 0, \ldots n$, sono i *nodi di interpolazione* e possono essere distribuiti arbitrariamente nell'intervallo anche se spesso, per motivi pratici, si preferisce una distribuzione uniforme cioè tale che

$$x_{k+1} - x_k = \Delta x, \text{ per } k = 1, \ldots, n-1 \tag{2.7}$$

in questo caso si dice che Δx è il passo di interpolazione.

È possibile ottenere una buona approssimazione del valore di f in un generico punto $x \in [a, b]$ dalla conoscenza dei valori $f(x_0), ..., f(x_n)$? Questo è il problema della interpolazione di una funzione che viene solitamente affrontato determinando un polinomio p_n (*polinomio interpolatore*) i cui valori coincidano con quelli di f nei punti $x_0, ..., x_n$ e calcolando poi $p_n(x)$. Naturalmente, la condizione

$$f(x_i) = p_n(x_i), i = 0, \ldots, n \tag{2.8}$$

non implica affatto che $f(x) = p_n(x)$ per $x \in [a, b]$. Quindi per risolvere in maniera soddisfacente il problema di interpolazione ci servirà una stima dell'errore

$$|f(x) - p_n(x)|, \quad \text{per ogni } x \in [a, b] \ .$$

che permetta di sapere che errore commettiamo sostituendo al valore della funzione nel punto x il corrispondente valore del polinomio. Osserviamo che, per costruzione, il polinomio interpolatore che verifica la (2.8) è unico. Infatti, se ci fosse un altro polinomio q_n che verifica la stessa condizione, avremmo

$$r_n(x_i) := p_n(x_i) - q_n(x_i) = 0, \quad \text{per ogni } i = 0, \ldots, n \tag{2.9}$$

e questo implica, per il teorema fondamentale dell'algebra, che il polinomio r_n è identicamente nullo dal momento che è un polinomio di grado n che si

annulla in $(n+1)$ punti distinti. L'unico polinomio di grado n che soddisfa la condizione (2.8) è detto *polinomio di Lagrange* e viene indicato come $\Pi_n(f)$.

Non ci resta allora che trovare un procedimento generale per costruire questo polinomio interpolatore e dimostrare la stima dell'errore. Cominciamo a trattare qualche caso particolare. Se f è nota in due soli punti x_0 e x_1, il polinomio interpolatore sarà di primo grado e potremo scriverlo nella forma

$$p_1(x) = Ax + B \ .$$

Le costanti A e B sono determinate dalle due condizioni di passaggio:

$$\begin{aligned} p_1(x_0) &= Ax_0 + B = f(x_0) \\ p_1(x_1) &= Ax_1 + B = f(x_1) \ . \end{aligned}$$

Da questo sistema lineare, con un semplice calcolo, si ottengono i coefficienti A e B e l'espressione del polinomio di Lagrange

$$\Pi_1(x) = f(x_0)\frac{(x - x_1)}{(x_0 - x_1)} + f(x_1)\frac{(x - x_0)}{(x_1 - x_0)} \ ,$$

E' immediato verificare che le condizioni di passaggio nei punti x_0 ed x_1 sono verificate.

Se f è nota in tre punti x_0, x_1, x_2, per determinare il suo polinomio interpolatore di secondo grado dovremmo risolvere un sistema lineare 3×3 per determinare i coefficienti. Nel caso generale poi dovremmo risolvere un sistema lineare di taglia $(n+1)$ che sappiamo essere un procedimento piuttosto costoso dl punto di vista computazionale: per un sistema di questo tipo (che non ha una struttura particolare) usando il metodo di Gauss costerebbe $O((n+1)^3)$ operazioni elementari.
Cerchiamo quindi un modo più diretto per costruire il polinomio di Lagrange e consideriamo, ad esempio, il caso $n = 2$. Osserviamo che, per costruirlo, basterebbe trovare tre polinomi di secondo grado nella forma

$$\begin{aligned} \ell_0(x) &= A_0x^2 + B_0x + C_0 \\ \ell_1(x) &= A_1x^2 + B_1x + C_1 \\ \ell_2(x) &= A_2x^2 + B_2x + C_2 \end{aligned}$$

tali che

$$\begin{aligned} \ell_0(x_0) &= 1 \quad \ell_0(x_1) = 0 \quad \ell_0(x_2) = 0 \\ \ell_1(x_0) &= 0 \quad \ell_1(x_1) = 1 \quad \ell_1(x_2) = 0 \\ \ell_2(x_0) &= 0 \quad \ell_2(x_1) = 0 \quad \ell_2(x_2) = 1 \ . \end{aligned} \tag{2.10}$$

È infatti immediato controllare che se ℓ_0, ℓ_1, ℓ_2 verificano queste condizioni, allora

$$\Pi_2(f)(x) = f(x_0)\ell_0(x) + f(x_1)\ell_1(x) + f(x_2)\ell_2(x) \tag{2.11}$$

è il polinomio interpolatore cercato e i polinomi ℓ_k, $k = 0, \ldots, 2$ si chiamano polinomi di base di Lagrange.

Per determinare le costanti A_i, B_i, C_i $(i = 0, 1, 2)$ osserviamo che le condizioni su ℓ_0 in (2.10) si traducono nel seguente sistema di 3 equazioni lineari nelle 3 incognite A_i, B_i, C_i:

$$\begin{cases} A_0x_0^2 + B_0x_0 + C_0 = 1 \\ A_0x_1^2 + B_0x_1 + C_0 = 0 \\ A_0x_2^2 + B_0x_2 + C_0 = 0 \ , \end{cases}$$

la cui matrice dei coefficienti ha determinante dato da:

$$d = (x_1 - x_2)[x_0^2 - x_0(x_1 + x_2) + x_1x_2] \neq 0 \ ,$$

dato che $x_0 \neq x_1 \neq x_2$. La matrice corrispondente a questi sistemi si chiama *matrice di Vandermonde*.

Risolvendo questo sistema si potrebbero calcolare A_0, B_0, C_0 e, in maniera analoga, si possono calcolare A_i, B_i, C_i $(i = 1, 2)$. La formula del polinomio interpolatore che si ottiene infine è la seguente,

$$\begin{aligned} \Pi_2(f)(x) &= f(x_0)\frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)} + f(x_1)\frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)} + \\ &+ f(x_2)\frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)}. \end{aligned} \tag{2.12}$$

Tuttavia questa strada diventa abbastanza complessa al crescere di n, inoltre i sistemi lineari corrispondenti risultano malcondizionati. Si preferisce quindi costruire i *polinomi di base di Lagrange* direttamente, osservando che la seguente definizione

$$\begin{aligned} \ell_k(x) &:= \frac{(x - x_0)\ldots(x - x_{k-1})(x - x_{k+1})\ldots(x - x_n)}{(x_k - x_0)\ldots(x_k - x_{k-1})(x_k - x_{k+1})\ldots(x_k - x_n)} = \\ &= \prod_{i=0, i\neq k}^{n} \frac{x - x_i}{x_k - x_i} \end{aligned} \tag{2.13}$$

garantisce che essi verificano le condizioni

$$\ell_k(x_i) = 0 \text{ per } i \neq k \text{ e } \ell_k(x_i) = 1 \text{ per } i = k, k = 0, \ldots, n. \tag{2.14}$$

Questo permette di ottenere una semplice rappresentazione del polinomio di Lagrange, che non richiede la soluzione di nessun sistema lineare per il calcolo dei coefficienti

$$\Pi_n(f)(x) := \sum_{k=0}^{n} f(x_k)\ell_k(x). \tag{2.15}$$

Si può dimostrare facilmente che i polinomi di base di Lagrange costituiscono una base nello spazio dei polinomi di grado n (esercizio 15).

Notiamo che per costruire il polinomio di Lagrange non è necessaria nessuna ipotesi di regolarità su f, basta avere i valori di f nei nodi di interpolazione. Da questo punto di vista, l'approssimazione di Lagrange è molto diversa dallo sviluppo di Taylor che fornisce una buona approssimazione vicino al centro dello sviluppo ma richiede la conoscenza delle derivate successive della funzione fino all'ordine n. Ovviamente, in questa generalità, non ci si può attendere che $\Pi_n(f)$ fornisca una buona approssimazione di f in punti diversi dai nodi come mostra il seguente esempio.

Esempio. Supponiamo di conoscere i valori $f(-1) = -100$ e $f(1) = 100$ di una funzione f definita su $[-1, 1]$. Il suo polinomio interpolatore di primo grado costruito sui nodi $x_0 = -1$ e $x_1 = 1$ è $\Pi_1(f)(x) = 100x$.

Se f fosse, ad esempio la funzione costante a tratti che vale -100 per $x \in [-1, 0)$ e 100 per $x \in [0, 1]$, si avrebbe un grande errore in $x = 0$, infatti

$$f(0) = 100, \text{ mentre } \Pi_1(f)(0) = 0 \ .$$

Questo avviene perchè la funzione ha una discontinuità in 0. Si osservi anche che, in questo caso, l'approssimazione è invece buona per x vicino a -1 oppure a 1 poichè lì la funzione è continua. Infatti,

$$f(0.99) = 100 \ , \qquad p(0.99) = 99 \ .$$

Osserviamo anche che la funzione per la funzione $f(x) = 50(x+1)^2 - 100$ si ha comunque $f(-1) = -100$ e $f(1) = 100$ ma in questo caso l'approssimazione darebbe in $x = 0$ risultati migliori, visto che $f(0) = -50$ verrebbe approssimato con $\Pi_1(f)(0) = 0$. Cosa accadrebbe se imponessimo anche la condizione di passaggio $\Pi_2(\overline{x}) = f(\overline{x})$ in un altro punto $\overline{x} \in (-1, 1)$? $\triangle$

Come vedremo adesso, un andamento "regolare" di f in $[a, b]$ elimina la patologia evidenziata nell'esempio. Si può anzi stimare l'errore commesso sostituendo a $f(x)$ il valore del suo polinomio interpolatore $\Pi_n(f)$. Si osservi che il teorema seguente non richiede alcuna ipotesi su come sono distribuiti gli x_k in $[a, b]$ ma richiede tutte le derivate di f fino all'ordine $n + 1$.

Teorema 2.1. *Se f è continua e dotata di derivate continue in $[a,b]$ fino all'ordine $n+1$, allora per ogni $x \in [a,b]$ si ha:*

$$|f(x) - \Pi_n(f)(x)| \leq \sup_{z\in[a,b]} |f^{(n+1)}(z)| \frac{(b-a)^{n+1}}{(n+1)!} \ . \tag{2.16}$$

Dimostrazione. Supponiamo, per semplificare le notazioni, che sia $n = 2$. La dimostrazione del caso generale segue le stesse linee e verrà lasciata per esercizio. Sia x un punto di $[a,b]$, diverso da tutti gli $x_i (i = 0,1,2)$ e consideriamo la funzione L definita per $t \in [a,b]$ da:

$$\begin{aligned} L(t) := f(t) - \Pi_2(f)(t) + \\ - \frac{f(x) - \Pi_2(f)(x)}{(x-x_0)(x-x_1)(x-x_2)} (t-x_0)(t-x_1)(t-x_2) \end{aligned} \tag{2.17}$$

Osserviamo che

$$L(x_0) = L(x_1) = L(x_2) = L(x) = 0 \ .$$

Supponendo che $x_0 < x_1 < x_2 < x$, per il teorema di Rolle esistono dunque

$$\xi_0 \in (x_0, x_1) \ , \quad \xi_1 \in (x_1, x_2) \ , \quad \xi_2 \in (x_2, x)$$

tali che

$$L'(\xi_i) = 0 \ , \quad i = 0,1,2 \ .$$

Per la stessa ragione esistono

$$\eta_0 \in (x_0, x_1) \ , \qquad \eta_1 \in (\xi_1, \xi_2) \ , \qquad \xi \in (\eta_0, \eta_1)$$

tali che

$$L''(\eta_i) = 0 \ , \qquad L'''(\zeta) = 0 \qquad i = 0,1.$$

D'altra parte, calcolando la derivata terza di L a partire da (2.17), si trova:

$$L'''(t) = f'''(t) - \Pi_2'''(f)(t) - 3! \frac{f(x) - \Pi_2(f)(x)}{(x-x_0)(x-x_1)(x-x_2)} \ .$$

Tenendo conto del fatto che $\Pi_2(f)$ è di secondo grado, si ha $\Pi_n'''(f)(t) \equiv 0$ e quindi la condizione $L'''(\zeta) = 0$ implica

$$f(x) - \Pi_2(f)(x) = \frac{f'''(\zeta)}{3!} (x-x_0)(x-x_1)(x-x_2) \ .$$

Maggiorando ogni termine $x - x_i$, $i = 0, 1, 2$, con $b - a$ si ottiene il risultato. □

Osserviamo che l'uso pratico della stima (2.16) richiede tuttavia un'analisi preliminare della funzione f e la conoscenza del valore $\sup_{z\in[a,b]} |f^{(n+1)}(z)|$, o per lo meno di una sua maggiorazione. Osserviamo anche che la (2.16), come del resto tutte le stime dell'errore che discuteremo in questo libro, dà informazioni sul "caso peggiore". Per particolari valori di x potrà accadere infatti che la differenza $|f(x) - \Pi_n(f)(x)|$ sia molto minore del numero a secondo membro della (2.16) (basti pensare che per $x = x_k$ essa è addirittura nulla e che, per continuità, in un intorno dei nodi l'errore sarà piccolo). La stima può anche essere utilizzata per trovare il numero minimo di nodi che garantisce un errore minore di una soglia fissata ε.

Esempio. Si vuole determinare il minimo numero di punti necessari per calcolare per interpolazione il valore di $f(x) = e^x$, per $x \in [0, 1]$, con un errore minore di $\varepsilon = 10^{-3}$. Si usa la stima (2.16), osservando che $\sup_{x\in[0,1]} f^{(n+1)}(x)| = e$, qualunque sia n. Si tratta quindi di trovare il minimo valore di n tale che

$$e/(n+1)! < 10^{-3} \ .$$

Dato che $10^3 \cdot e < 3142$, basterà prendere $n = 6$, dal momento che $6! = 720$ mentre $7! = 5040$. △

Ci chiediamo allora che la stima dell'errore appena dimostrata sia sufficiente a garantire una buona approssimazione della funzione f al crescere del numero dei nodi di interpolazione. Ci chiediamo cioè se l'errore indicato della stima tenda a 0 per n che tende a infinito. Osservando bene la stima si vede che l'errore dipende dalla derivata $(n+1)$-esima della funzione f e da una frazione dove a numeratore abbiamo una costante elevata alla n ed a denominatore il fattoriale di n. Poichè la velocità con cui diverge il fattoriale è più alta di qualunque potenza C^n (provate a dimostrarlo), se avessimo una stima del tipo

$$\sup_x |f^n| < D_{max}, \text{ per ogni } n \tag{2.18}$$

potremmo allora concludere che l'approssimazione migliora per un numero di nodi crescente dal momento che avremmo la maggiorazione dell'errore seguente

$$|f(x) - \Pi_n(f)(x)| \leq D_{max} \frac{(b-a)^{n+1}}{n!} \tag{2.19}$$

ed il termine a destra tende a 0 per n che tende ad infinito.

Tuttavia la condizione è molto restrittiva dal momento che chiede che tutte le derivate siano uniformemente maggiorate da una costante ed in generale non è verificata. In particolare, allo scopo di ottenere un controesempio, possiamo osservare che un polinomio di grado n verifica all'infinito la condizione

$$\lim_{|x|\to+\infty} |p_n(x)| = +\infty \tag{2.20}$$

dunque non potrà approssimare accuratamente una funzione che abbia un asintoto orizzontale. Ad esempio, se consideriamo la *funzione di Runge* $f(x) = 1/(1+x^2)$ che verifica

$$\lim_{|x|\to+\infty} f(x) = 0 \tag{2.21}$$

in un intervallo simmetrico rispetto all'origine (ad esempio [-4,4]) si osserva che al crescere di n l'approssimazione di Lagrange va malissimo al di fuori dell'intervallo [1.6,1.6] proprio perchè le oscillazioni del polinomio di Lagrange aumentano con n. Questa osservazione motiva la costruzione di una interpolazione di Lagrange composita che affronteremo nel prossimo paragrafo.

2.5 Interpolazione di Lagrange composita

La difficoltà che abbiamo incontrato nel paragrafo precedente è legata al fatto che al crescere del numero dei nodi cresce il grado del polinomio interpolatore. E' allora conveniente per avere un controllo dell'errore scegliere un numero di nodi di interpolazione basso per tenere basso il grado del polinomio, questo risultato si può ottenere facendo una ricostruzione polinomiale locale. Questa tecnica si chiama *interpolazione di Lagrange composita* e produce una funzione approssimante *polinomiale a tratti* (non un unico polinomio come nel caso visto nel paragrafo precedente).

2.5.1 Interpolazione lineare composita

Vediamo, per cominciare, come costruire una funzione polinomiale a tratti in cui la ricostruzione locale sia di grado 1. Il modo più semplice consiste nel dividere l'intervallo $[a, b]$ in N sotto–intervalli, scegliendo $N+1$ nodi che per semplicità supporremo equidistanti. La griglia sarà quindi data dai nodi

$$a = x_0 < x_1 < x_2 < .... < x_N = b$$

con

$$x_{k+1} - x_k = (b-a)/N \ , \qquad i = 0, 1, ..., N-1$$

Se consideriamo i nodi a due a due, potremo ricostruire in ogni intervallo $I_k := [x_k, x_{k+1}]$ il polinomio di grado 1 che unisce i due punti sul grafico $(x_k, f(x_k))$ ed $(x_{k+1}, f(x_{k+1}))$. Questo tecnica si chiama *interpolazione lineare composita* e può essere utilizzata in presenza di un numero di nodi arbitrariamente grande. E' molto facile da implementare ed è abbastanza accurata se la funzione non presenta forti oscillazioni.
Ricaviamo ora una stima dell'errore. Consideriamo la funzione $\widehat{f}$ che si ottiene interpolando linearmente i valori della f sui nodi. La definizione di $\widehat{f}$ nell'intervallo $I_k := [x_k, x_{k+1})$, $k = 0, 1, \ldots, N$ è dunque:

$$\widehat{f}(x) \quad := \quad f(x_k) + \frac{f(x_{k+1}) - f(x_k)}{h}(x - x_k), \ x \in I_k \tag{2.22}$$

dove $h := (b-a)/N$ è il passo di interpolazione (spesso indicato anche come Δx). Osserviamo che per costruire questa griglia uniforme basterà definire i nodi attraverso la formula

$$x_k := a + kh$$

che riduce la propagazione dell'errore nel calcolo dei nodi rispetto alla formula ricorsiva $x_{k+1} = x_k + h$.

Vogliamo stimare l'errore che si commette sostituendo ad $\widehat{f}$ la funzione $\widehat{f}$. A questo scopo, supporremo che f sia derivabile due volte in (a, b) e che f'' sia limitata, cioè che

$$\sup_{\xi \in [a,b]} |f''(\xi)| \leq M \tag{2.23}$$

per ogni $x \in (a, b)$. Tenendo conto del fatto che, per costruzione, $f(x_i) = \widehat{f}(x_i)$, si deduce facilmente dal teorema di Lagrange che la seguente relazione

$$|f(x) - \widehat{f}(x)| = |f'(\zeta_k) - \widehat{f}'(x_k)| \ |x - x_k| \tag{2.24}$$

è valida per ogni $x \in [x_k, x_{k+1})$.

Sostituendo allora nella (2.24) la forma esplicita di $\widehat{f}'$, che si calcola facilmente a partire da (2.22), si ottiene

$$|f(x) - \widehat{f}(x)| = |f'(\xi_k) - \frac{f(x_{k+1}) - f(x_k)}{h}|x - x_k| \ . \tag{2.25}$$

Poiché, sempre per il teorema di Lagrange,

$$f(x_{k+1}) - f(x_k) = f'(\eta_k)(x_{k+1} - x_k) = f'(\eta_k)h \ , \tag{2.26}$$

con $\eta_i \in (x_k, x_{k+1})$, si ottiene che per ogni $x \in [x_k, x_{k+1})$ vale la disuguaglianza

$$|f(x) - \widehat{f}(x)| = |f'(\xi_k) - f'(\eta_k)|\ |x - x_k| \leq |f'(\xi_k) - f'(\eta_k)|h\ . \qquad (2.27)$$

Finalmente, riapplicando il teorema di Lagrange alla f', da (2.27) segue:

$$|f(x) - \widehat{f}(x)| \leq \sup_{\xi \in (x_k, x_{k+1})} |f''(\xi)|h^2 \leq Mh^2\ , \qquad (2.28)$$

per ogni $x \in [x_k, x_{k+1})$. Poichè l'intervallo I_k è generico, per avere una stima dell'errore valida su tutto l'intervallo $[a, b]$ basterà prendere il massimo della derivata seconda di f. La stima complessiva è quindi

$$|f(x) - \widehat{f}(x)| \leq \sup_{\xi \in (a,b)} |f''(\xi)|h^2 \leq Mh^2\ ,\ \text{per } x \in [a, b]. \qquad (2.29)$$

2.5.2 Interpolazione quadratica composita

Il modo più semplice per costruire una interpolazione composita di grado 2 è quello di usare anche i punti medi dei sotto intervalli I_k. Costruita una griglia di $N + 1$ nodi, $x_0 \ldots, x_N$, aggiungiamo come nodi di interpolazione i punti medi $m_k := (x_k + x_{k+1})/2$ in questo modo avremo complessivamente $2N + 1$ nodi di interpolazione ma il grado della ricostruzione polinomiale a tratti sarà sempre pari a 2 (e non salirà a $2N$ come sarebbe avvenuto se avessimo considerato il polinomio di Lagrange basato su tutti i nodi).
Conviene allora esaminare la situazione in un generico sotto intervallo I_k per ricavare la stima dell'errore.
La ricostruzione locale nell'intervallo I_k corrisponde alla parabola passante per i tre punti sul grafico $(x_k, f(x_k))$, $(m_k, f(m_k))$ ed $(x_{k+1}, f(x_{k+1}))$, cioè al polinomio $\Pi_2(f)$ basato sui tre nodi x_k, m_k, x_{k+1}. Avremo allora

$$\begin{aligned} \widehat{f}(x) &= \Pi_2(f)(x) = f(x_k)\frac{(x - m_k)(x - x_{k+1})}{(x_k - m_k)(x_k - x_{k+1})} + \qquad (2.30) \\ &+ f(m_k)\frac{(x - x_k)(x - x_{k+1})}{(m_k - x_k)(m_k - x_{k+1})} + f(x_{k+1})\frac{(x - x_k)(x - m_k)}{(x_{k+1} - x_k)(x_{k+1} - m_k)}. \end{aligned}$$

per ogni $x \in I_k$. Dal Teorema 2.1 otteniamo la stima dell'errore locale

$$|f(x) - \widehat{f}(x)| \leq \sup_{z \in I_k} |f'''(z)|\frac{h^3}{3!} \text{ per } x \in I_k \qquad (2.31)$$

che si può estendere immediatamente a tutto l'intervallo maggiorando la derivata terza su $[a, b]$. In conclusione, otteniamo

$$|f(x) - \widehat{f}(x)| \leq \sup_{\xi \in (a,b)} |f'''(\xi)| \frac{h^3}{3!} \leq M \frac{h^3}{3!} \ , \text{ per } x \in [a, b]. \tag{2.32}$$

Sulla base degli stessi argomenti, possiamo generalizzare la stima al caso di una *interpolazione composita di grado n in ciascuno dei sottointervalli* I_k. Evidentemente occorrerà considerare $n+1$ nodi di interpolazione in ciascuno dei sottointervalli I_k ed utilizzare come ricostruzione locale dell f in I_k il corrispondente polinomio di Lagrange $\Pi_n(f)$, definendo $\widehat{f}(x) = \Pi_n(f)(x)$, per $x \in I_k$. Ricordando la stima generale per l'errore della interpolazione di Lagrange avremo allora

$$|f(x) - \Pi_n(f)(x)| \leq \sup_{\xi \in I_k} |f^{(n+1)}(\xi)| \frac{h^{(n+1)}}{(n+1)!} \ , \text{ per } x \in I_k. \tag{2.33}$$

da cui ricaviamo la stima complessiva

$$|f(x) - \widehat{f}(x)| \leq \sup_{\xi \in (a,b)} |f^{(n+1)}(\xi)| \frac{h^{(n+1)}}{(n+1)!} \leq M \frac{h^{(n+1)}}{(n+1)!} \ , \text{ per } x \in [a, b]. \tag{2.34}$$

dove M questa volta è una maggiorazione della derivata $(n+1)$-esima di f.

2.6 Sviluppo di Taylor

Una diversa maniera di calcolare in modo approssimato il valore di una funzione in un punto assegnato x è quello di sostituire $f(x)$ con il valore di un suo polinomio di Taylor in x. Anche in questo caso si tratta di sostuire al calcolo della funzione quello di un polinomio approssimante ma la costruzione del polinomio di Taylor è completamente diversa da quella del polinomio di Lagrange. Lo sviluppo di Taylor all'ordine n è dato infatti da

$$T_n(f)(x) := \sum_{k=0}^{n} \frac{f^{(k)}(x_0)}{k!} (x - x_0)^k \tag{2.35}$$

dove il punto x_0 è il centro dello sviluppo ed $f^{(k)}$ indica la derivata k-esima della funzione f. Il polinomio di grado n è quindi costruito in modo da utilizzare tutte le derivate di f in x_0 fino all'ordine n, si avrà anche per costruzione

$$T_n^{(k)}(x_0) = f^{(k)}(x_0), \text{ per } k = 0, \ldots, n. \tag{2.36}$$

E' quindi necessario supporre che f abbia tutte le derivate fino alla n-esima nel centro dello sviluppo per poter scrivere il polinomio. Al contrario, nel polinomio di Lagrange servono solo i valori in $n+1$ punti distinti e per scrivere il polinomio non è richiesta nessuna regolarità (che però viene richiesta nella stima dell'errore). L'errore che si commette nella approssimazione di Taylor è dato dal resto

$$R_n(x, x_0) = f(x) - T_n^{(k)}(x_0) \tag{2.37}$$

e, come è noto, nel caso in cui f sia derivabile fino all'ordine $n+1$, si ha la seguente forma per il resto

$$R_n(x, x_0) = \frac{f^{(n+1)}(\xi)}{(n+1)!}(x - x_0)^{n+1} \tag{2.38}$$

(*resto in forma di Lagrange*). Da questa forma si deduce facilmente che il resto tende a 0 per n che tende a infinito almeno in un intorno di x_0. Osserviamo infatti che il resto diminuisce scegliendo un centro x_0 vicino ad x e questa osservazione suggerisce una strategia per ridurre il numero dei termini necessari ad avere una buona approssimazione della funzione f.

Illustriamo questo procedimento su alcuni semplici esempi.

Se $f(x) = e^x$, allora per ogni $x \in \mathbb{R}$ e per ogni intero positivo n, lo sviluppo di Taylor della funzione esponenziale con punto iniziale $x_0 = 0$ è :

$$e^x = 1 + x + \frac{x^2}{2} + \frac{x^3}{3!} + ... + \frac{x^n}{n!} + R_n(x, 0) \ . \tag{2.39}$$

Per l'errore $R_n(x, 0)$, con $x > 0$, si ha la seguente stima:

$$\frac{x_{n+1}}{(n+1)!} \leq R_n(x, 0) \leq e^{[x]+1}\frac{x^{n+1}}{(n+1)!} < 3^{[x]}3\frac{x^{n+1}}{(n+1)!} \tag{2.40}$$

Il calcolo approssimato di e^x può essere quindi effettuato riconducendolo a quello elementare di $T_n(x) = \sum_{k=0}^{n} x^k/k!$, a meno di un errore, dipendente da x (e da x_0), valutabile tramite la stima (2.38).

Esempio. Il valore di $e^{0.4}$ con tre cifre decimali esatte è dato da

$$\sum_{k=0}^{5} \frac{(0.4)^k}{k!} \ ,$$

dal momento che che $n = 5$ verifica la diseguaglianza $3 \cdot (0.4)^{n+1}/(n+1)! < 0.5 \cdot 10^{-4}$. In generale, se $x > 0$ è sufficientemente piccolo, basterà un

polinomio di grado abbastanza basso (e quindi un numero non elevato di operazioni algebriche) per ottenere una buona approssimazione di e^x. La situazione diventa assai meno favorevole per grandi valori di x. Infatti, pur essendo vero che

$$\lim_{n\to+\infty} 3 \cdot 3^{[x]} \frac{x^{n+1}}{(n+1)!} = 0 \ , \qquad \text{qualunque sia } x \ ,$$

il minimo valore di n che verifica la disuguaglianza

$$3 \cdot 3^{[x]} \frac{x^{n+1}}{(n+1)!} < \varepsilon$$

diventa rapidamente grande al crescere di x (vedi esercizio 7). Osserviamo infine che c'è un modo più astuto per calcolare e^x quando x è grande. Possiamo scrivere sempre $e^x = e^{[x]+x-[x]} = e^{[x]}e^{x-[x]}$, in questo modo per calcolare il secondo termine basteranno pochi termini dello sviluppo di Taylor centrato in 0 (perché $x - [x]$ è minore di 1) mentre per il calcolo del primo termine basterà elevare e ad una potenza intera (e può essere ottenuto ancora tramite lo sviluppo di Taylor oppure tramite un metodo di ricerca degli zeri). △

Per quanto riguarda la funzione $\sin x$, l'approssimazione basata sullo sviluppo di Taylor non presenta le stesse difficoltà evidenziate per la funzione esponenziale. Infatti, utilizzando le formule

$$\sin(x + 2\pi) = \sin x$$

$$\sin(-x) = -\sin x \ ,$$

che indicano la periodicità della funzione ed il fatto che la funzione è dispari, sarà sempre possibile riportarsi ad un intorno dell'origine relativamente piccolo. Basterà allora considerare lo sviluppo

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} + ... + (-1)^{n-1} \frac{x^{2n-1}}{(2n-1)!} + R_{2n}(x, 0) \ ,$$

per $x \in [0, \pi]$. L'errore è stimato da

$$|R_{2n}(x)| \leq \frac{|x|^{2n+1}}{(2n+1)!} \ .$$

Si ossservi anche che nello sviluppo del seno i termini di grado pari scompaiono perchè le corrispondenti derivate sono nulle in 0. Analoghe considerazioni possono essere fatte per le funzioni $\cos x$ e arctang x.

2.7 Esercizi

Esercizio 2.1. Supponiamo che per un calcolatore sia $M = 2$ ed $E = 1$. Quanti sono i numeri macchina? Quali sono il minimo ed il massimo di tali numeri?

Esercizio 2.2. Supponiamo che un calcolatore abbia $M = 4$ ed $E = \infty$. Calcolare $rd(x)$ per i seguenti valori di x:

$$\begin{array}{lll} 0.12345216 & 100.31451 & 2.34521343 \\ 125.3216 & 1.000024 & 0.0000157\ . \end{array}$$

Esercizio 2.3. Si considerino le seguenti espressioni:

$$\begin{array}{ll} A = (N+1)^{1/2} - N^{1/2} & B = 1/[(N+1)^{1/2} + N^{1/2}] \\ C = N^{1/2}[(1+1/N)^{1/2} - 1] & D = 1/2N^{1/2}\ . \end{array}$$

Dimostrare che per ogni numero positivo N si ha $A = B = C$; scrivere un programma che per ogni N in input calcoli e scriva sullo schermo i corrispondenti valori di A, B, C, D e provarlo per numerosi valori di N (ad esempio, $N = 10, 100, 1000...$). Confrontare, in particolare, i valori di A e D corrispondenti a $N = 10^{-2}$ e $N = 10^3$).

Esercizio 2.4. Scrivere un programma che sommi i numeri $1/N^3$ ($N = 1, 2, ..., 500$) prima nell'ordine crescente e poi in quello decrescente. Confrontare i due risultati.

Esercizio 2.5. Confrontare i valori di $1+1/N$ e $(N+1)/N$ per varie scelte di N (ad esempio, $N = 10, 100, 1000, ...$). Riflettere sui risultati numerici ottenuti.

Esercizio 2.6. Il calcolo in virgola mobile di $a_1 + a_2 + ... + a_n$ può dare luogo ad un errore relativo arbitrariamente grande in dipendenza da n (vedi la formula (2.28)). Se tutti gli addendi a_i hanno lo stesso segno tale errore è viceversa limitato. Darne una maggiorazione.

Esercizio 2.7. Usando le formule (2.39) e (2.40) determinare n in modo tale che

$$e^3 - \sum_{k=0}^{n} \frac{x^k}{k!} < 0.5\ 10^{-4}\ .$$

Esercizio 2.8. Valutare, usando la (2.28), gli errori relativi nel calcolo di x^n, $1/x$ e $x^{1/n}$.

Esercizio 2.9. Implementare un algoritmo basato sullo sviluppo di Taylor che calcoli $\sin x$ con tre cifre decimali esatte. Provarlo per $x = 0.01$, $x = 0.1$, $x = 0.5$, $x = 10$, $x = 100$ e confrontare i risultati con quelli forniti dalla funzione $\sin(x)$ del vostro calcolatore.

Esercizio 2.10. Ripetere l'esercizio 9 per le funzioni $\cos x$ e arctang x.

Esercizio 2.11. Scrivere un programma che implementi l'algoritmo (A1) e quello di Ruffini-Horner. Calcolare con i due metodi i valori dei seguenti polinomi nel punto $x = 0.5$:

$$\begin{array}{ll} p(x) = 10x^2 + 14x + 5 & p(x) = 33x^9 + 7x^3 + 0.1 \\ p(x) = 81x^7 + 5x^6 + 4x^3 & p(x) = 3x + 2 \ . \end{array}$$

Esercizio 2.12. Implementare l'algoritmo di interpolazione quadratica di una funzione f su $[0, 1]$, combinato con l'algoritmo di Ruffini–Horner. Provarne il funzionamento sui seguenti esempi:

$$\begin{array}{ll} (a) f(x) = x^2 & (b) f(x) = e^x \\ (c) f(x) = \sin x & (d) f(x) = \log(x + 1) \\ (e) f(x) = x^2/(1 + x2) & (f) f(x) = \log(x + 1)/e^x \ , \end{array}$$

tabulando i valori di f e della sua interpolata nei punti $x = 0.25$, $x = 0.5$, $x = 0.75$.

Esercizio 2.13. Calcolare i polinomi di Taylor di ordine 2 delle funzioni dell'esercizio 12 e, usando l'algoritmo di Ruffini–Horner, calcolare valori approssimati di tali funzioni nei punti $x = 0.25$, $x = 0.5$, $x = 0.75$. Confrontare con i risultati dell'esercizio 12.

Esercizio 2.14. Dimostrare la stima dell'errore per il polinomio di Lagrange nel caso generale con $n + 1$ nodi di interpolazione. *Suggerimento: costruire la nuova funzione L tenendo conto dei nodi di interpolazione ed iterare nella applicazione del teorema di Rolle n volte.*

Esercizio 2.15. Dimostrare che i polinomi di base di Lagrange sono una base nello spazio dei polinomi di grado n, $\mathbb{P}_n$.