CORE
Metadata, citation and similar papers at core.ac.uk
Provided by Electronic Thesis and Dissertation Archive - Università di Pisa

UNIVERSITÀ DEGLI STUDI DI PISA

FACOLTÀ DI SCIENZE MATEMATICHE FISICHE E NATURALI

Corso di Laurea in Matematica

TESI DI LAUREA

Metodi per il calcolo di funzioni elementari in alta precisione

Candidato
Gabriele Stilli

Relatore
Prof. Dario A. Bini

Controrelatore
Prof. Luca Gemignani

Anno Accademico 2004/2005

Questo lavoro è stato prodotto utilizzando esclusivamente Software Libero. La composizione è stata realizzata con il sistema LaTeX, usando gli editor di testo Jed ed Emacs su sistema operativo GNU/Linux (distribuzioni Debian e Red Hat).

Indice

Prefazione

Meglio essere ottimisti e avere torto, che pessimisti e avere ragione.

Albert Einstein

I metodi di risoluzione numerica di problemi matematici vengono solitamente implementati in un ambiente di calcolo caratterizzato da una aritmetica in virgola mobile (aritmetica *floating point*). Questo fa sì che i piccoli errori generati in ogni singola operazione aritmetica svolta in precisione finita possano accumularsi nel corso del calcolo e possano essere amplificati fino al punto da rendere inutilizzabile il risultato effettivamente calcolato.

Questo è un problema classico dell'analisi numerica che viene affrontato mediante lo studio della stabilità degli algoritmi (cioè della proprietà dell'algoritmo di non amplificare gli errori generati nei calcoli) e mediante l'analisi del condizionamento del problema cioè della sensibilità che ha il risultato rispetto a "piccole" perturbazioni dei dati.

Certamente il calcolo accurato può essere svolto mediante la scelta di un algoritmo numericamente stabile, purché il problema sia ben condizionato. Però nel caso di problemi mal condizionati, poiché i soli inevitabili errori di rappresentazione dei dati producono elevati errori nel risultato qualunque

sia l'algoritmo usato, gli unici rimedi possibili per calcolare una soluzione attendibile del problema sono:

a) svolgere se possibile i calcoli in modo simbolico così da eliminare gli errori sia nella rappresentazione dei dati che nello svolgimento delle operazioni floating point;

b) adottare una aritmetica con precisione elevata e possibilmente modificabile dinamicamente nel corso dei calcoli.

La prima delle due opzioni in generale produce complessità di calcolo che crescono esponenzialmente col numero di operazioni svolte per cui, nella pratica, il rimedio più conveniente è quello dato nel punto b).

Alla fine degli anni '70, Richard Brent [1] introdusse uno dei primi pacchetti software in Fortran 77 con una aritmetica in precisione variabile. Questo pacchetto includeva il calcolo delle funzioni elementari (esponenziale, logaritmo, radici, funzioni trigonometriche) in alta precisione. Successivamente altri pacchetti più sofisticati sono stati presentati in letteratura, fra questi il pacchetto MPFUN di David Bailey [2] scritto inizialmente in Fortran 77 e poi portato in Fortran 90 e il più sofisticato pacchetto GMP della GNU scritto in C [3].

Nella realizzazione e analisi di algoritmi per il calcolo di funzioni in alta precisione, gioca un ruolo importante il concetto di complessità e la scelta dell'algoritmo di calcolo assume un aspetto cruciale per l'efficienza computazionale del pacchetto. In letteratura sono apparsi diversi contributi relativi al disegno e analisi di metodi di bassa complessità per il calcolo di funzioni elementari. In particolare è da sottolineare l'importante contributo dato nel 1971 da A. Schönhage e V. Strassen [4] nel definire un metodo per la moltiplicazione di interi di n bit in $O(n \log n \log\log n)$ operazioni binarie anziché

$O(n^2)$ del metodo classico. Storicamente importante è anche il lavoro di R. Brent [5], pubblicato nel 1976, che dimostra come si possano calcolare esponenziale, logaritmo, funzioni trigonometriche e trigonometriche inverse con un errore relativo di $O(2^{-n})$, cioè con n cifre di precisione, in un tempo di $O(t_n(M) \log n)$ operazioni binarie (dove $t_n(M)$ è il tempo richiesto per l'effettuazione di una moltiplicazione di interi di n bit), sfruttando il metodo di Newton, gli integrali ellittici e il metodo della media aritmetico-geometrica.

Più recentemente, Smith [6] ha sviluppato un algoritmo per il calcolo di esponenziale e funzioni trigonometriche, basato sulla somma dei termini delle loro serie di Taylor opportunamente riordinati e raggruppati, in un tempo di $O(n^{1/3}t_n(M))$, quindi asintoticamente più lento di quello di Brent, ma più efficiente di quest'ultimo per precisioni di poche centinaia di cifre. L'algoritmo di Smith è stato migliorato nel 1994 da Xu Guo-liang e Li Jiakai [7] per un fattore moltiplicativo di circa $(\frac{24}{7})^{1/3} \approx 1.5$ tramite l'uso delle approssimanti di Padé delle funzioni da calcolare al posto delle loro serie di Taylor. I metodi qui descritti sono ad oggi i più veloci conosciuti per l'effettuazione dei calcoli in questione.

In questa tesi si raccolgono in modo sistematico i metodi principali presenti in letteratura e se ne dà una descrizione in forma algoritmica in modo che la traduzione di ciascun metodo in termini di programma in linguaggio di alto livello sia facilmente realizzabile.

Nel primo capitolo introduciamo gli strumenti generali utili per la nostra analisi. In particolare, presentiamo alcuni fra gli algoritmi concretamente più efficienti per la moltiplicazione fra interi (tale operazione gioca un ruolo importante nello sviluppo dei successivi algoritmi di calcolo): l'algoritmo di Karatsuba e quello basato sulla trasformata discreta di Fourier (DFT, Discrete Fourier Transform), calcolata con il metodo FFT (Fast Fourier Trans-

form). Tali algoritmi vengono estesi in modo naturale alle operazioni di numeri floating point. Quindi diamo la definizione di convergenza di un metodo iterativo ed esaminiamo le proprietà di convergenza del metodo di Newton, che di questi è uno dei più efficienti e che servirà poi a costruire gli algoritmi più efficaci per il calcolo del quoziente, della radice quadrata e cubica e, in generale, dell'inversa di una qualsiasi funzione sufficientemente regolare. Il capitolo si conclude con la presentazione di alcuni strumenti utili agli algoritmi presentati nei capitoli successivi: la tecnica di esponenziazione binaria, su cui si basa anche un semplice algoritmo per il calcolo della radice n-esima di un numero reale, la formula di Stirling per l'approssimazione del fattoriale di un numero intero, il metodo di Ruffini-Horner per il calcolo del valore di un polinomio in un punto e la definizione di serie ipergeometrica.

Nel secondo capitolo, trattiamo alcuni metodi per il calcolo rapido in alta precisione della funzione esponenziale. Cominciamo con l'esame del calcolo dell'esponenziale tramite somma della sua serie di Taylor, quindi raffiniamo questo procedimento, prima utilizzando l'algoritmo presentato da Smith e poi presentando il miglioramento introdotto da Xu Guo-Liang e Li Jia-kai tramite l'uso delle approssimanti di Padé al posto della serie di Taylor. L'ultimo algoritmo che prendiamo in considerazione è quello dellla media aritmetico-geometrica (AGM, Arithmetic-Geometric Mean) di R. Brent.

Nel terzo capitolo, analizziamo i metodi per il calcolo della funzione logaritmo, la maggior parte dei quali derivano direttamente da quelli del calcolo dell'esponenziale visti in precedenza. Anche qui cominciamo prendendo in considerazione la serie di Taylor del logaritmo; quindi trattiamo il calcolo del logaritmo tramite l'inversione dell'esponenziale (ossia, la soluzione dell'equazione $\exp(x) - y = 0$ per ricavare $x = \log y$) ottenuta mediante il metodo di Newton ed infine descriviamo l'applicazione del metodo AGM al calcolo del

logaritmo.

Nel quarto capitolo, infine, i metodi già esposti in precedenza vengono applicati al calcolo delle funzioni trigonometriche e trigonometriche inverse. Viene anche presentato l'algoritmo CORDIC (COordinate Rotation DIgital Computer), utile per l'implementazione in ambienti in cui non è disponibile un moltiplicatore hardware integrato.

Capitolo 1

Introduzione e strumenti di calcolo

If you have an apple and I have an apple and we exchange apples, then you and I will still each have one apple. But if you have one idea and I have one idea and we exchange these ideas, then each of us will have two ideas.

George Bernard Shaw

Nel calcolo delle funzioni elementari in alta precisione, sono cruciali alcuni problemi di base che vengono esaminati in questo capitolo. Nell'ordine, tratteremo gli algoritmi per il calcolo del prodotto di interi, il metodo di Newton applicato al calcolo del reciproco di un numero in virgola mobile e al calcolo della radice n-esima di un numero in virgola mobile. Il capitolo si conclu-

de con la descrizione della tecnica di esponenziazione binaria, il metodo di Horner, la formula di Stirling e le serie ipergeometriche.

1.1 Moltiplicazione di interi

Sia B un intero maggiore di 1 e a un intero positivo e si consideri la rappresentazione in base B di a:

$$a = a_n a_{n-1} \dots a_1 a_0$$

per cui

$$a = \sum_{j=0}^{n} B^j a_j, \quad 0 \leq a_j < B.$$

Associamo ad a il polinomio $a(x) = \sum_{j=0}^{n} a_j x^j$.

Sia ora r un numero razionale e si consideri la sua rappresentazione in base B:

$$r = r_n r_{n-1} \dots r_1 r_0 . r_{-1} \dots r_{-m+1} r_{-m}$$

per cui

$$r = \sum_{j=-m}^{n} B^j r_j, \quad 0 \leq r_j < B.$$

Associamo ad r il polinomio (detto *di Laurent*) $r(x) = \sum_{j=-m}^{n} r_j x^j$.

Si osservi che, se a, b e c sono tre interi tali che $c = ab$ e se $a(x)$, $b(x)$ e $c(x)$ sono i polinomi associati rispettivamente ai tre interi, si ha subito $c(B) = a(B)b(B)$.

Definizione 1 *Siano dati due vettori di lunghezza n, $a = (a_0, a_1, \dots, a_{n-1})$ e $b = (b_0, b_1, \dots, b_{n-1})$. Si definisce* convoluzione *di a e b il vettore $c = (c_0, c_1, \dots, c_{2n-2})$ le cui componenti sono:*

$$c_k = \sum_{\substack{i,j=0 \\ i+j=k}}^{n-1} a_i b_j, \quad 0 \leq k \leq 2n-2.$$

Siano m e n i gradi rispettivamente di $a(x)$ e $b(x)$ e sia $\hat{c}(x) = a(x)b(x) = \sum_{j=0}^{m+n} \hat{c}_j x^j$ il polinomio prodotto di $a(x)$ e $b(x)$. È noto che i coefficienti di $\hat{c}(x)$ si ottengono facendo la convoluzione dei due vettori che hanno per componenti rispettivamente i coefficienti di $a(x)$ e $b(x)$.

In generale i coefficienti di $\hat{c}(x)$ non sono compresi fra 0 e $B-1$; di conseguenza, $c_j \neq \hat{c}_j$. È comunque facile scrivere le relazioni ricorsive che legano i c_j e i $\hat{c}_j$:

$$\begin{cases} c_0 = \hat{c}_0 \bmod B \\ c_j = (\hat{c}_j + (c_{j-1} \div B)) \bmod B \quad 1 \leq j \leq m+n, \end{cases} \tag{1.1}$$

dove con $x \div y$ si è indicato il quoziente della divisione intera di x per y, mentre con $x \bmod y$ il resto della stessa divisione intera.

Queste proprietà permettono di calcolare le cifre del prodotto di a e b attraverso il calcolo dei coefficienti di $\hat{c}(x)$ col seguente algoritmo:

1. si calcolano tramite convoluzione i coefficienti di $\hat{c}(x)$;

2. applicando le (1.1), si calcolano i c_j a partire dai $\hat{c}_j$.

In modo analogo si può procedere per la moltiplicazione di numeri razionali, in quanto i polinomi di Laurent si possono ridurre a polinomi classici tramite moltiplicazione per x^k, per un opportuno valore di $k \in \mathbb{N}$.

La moltiplicazione di due numeri di n cifre con il metodo ordinario richiede dunque n^2 moltiplicazioni di cifre ed altrettante addizioni di cifre. Ad esempio, il calcolo del prodotto di due numeri con un milione di cifre in base 10 ciascuno richiederebbe circa 10^{12} moltiplicazioni semplici; su una macchina capace di eseguire 10 milioni di operazioni al secondo, questo calcolo richiederebbe circa 28 ore. È intuitivo, perciò, che la moltiplicazione sia un'operazione particolarmente dispendiosa e che uno dei principali problemi del

calcolo delle funzioni in alta precisione sia quello di diminuire sia il tempo di calcolo di ogni singola moltiplicazione sia il numero di moltiplicazioni totali di un algoritmo, anche a costo di aumentare quello di altre operazioni più veloci, quali addizioni e sottrazioni.

Nei prossimi paragrafi vedremo come è possibile accelerare sensibilmente questi calcoli.

1.1.1 Il metodo di Karatsuba

Uno dei metodi migliori per accelerare la moltiplicazione di numeri con molte cifre è il metodo di Karatsuba. Questo metodo consiste nello scrivere i due fattori come somma di due parti, una composta dalle cifre di ordine inferiore e l'altra da quelle di ordine superiore. Quindi, per effettuare la moltiplicazione, si riscrivono i prodotti parziali in modo da minimizzare il numero di moltiplicazioni a precisione piena.

Siano dunque a e b i due numeri da moltiplicare e, per comodità, supponiamo che questi abbiano la stessa lunghezza in cifre, n, in una certa base B (se così non fosse, basta completare il numero più corto aggiungendo degli zeri all'inizio). Supponiamo inoltre che n sia pari (se non lo fosse, la parte più significativa del numero conterrà una cifra in meno); inoltre poniamo per comodità $T = B^{n/2}$. Spezziamo i numeri a e b in due parti uguali:

$$a = a_0 + a_1 T$$
$$b = b_0 + b_1 T.$$

Per moltiplicare a e b, l'algoritmo classico è quello di svolgere i prodotti nella maniera usuale:

$$ab = a_0 b_0 + T(a_0 b_1 + b_0 a_1) + T^2 a_1 b_1.$$

Ciò richiede 4 moltiplicazioni a precisione dimezzata (i quattro $a_i b_j$, che sono prodotti di numeri di lunghezza la metà degli originali). La relazione utilizzata dall'algoritmo di Karatsuba è invece:

$$ab = (1+T)a_0b_0 + T(a_1 - a_0)(b_0 - b_1) + (T + T^2)a_1b_1, \qquad (1.2)$$

che richiede solo 3 moltiplicazioni a precisione dimezzata. Si ha quindi un risparmio di una moltiplicazione lunga a prezzo del raddoppio (circa) del numero di addizioni e sottrazioni, cosa che è comunque vantaggiosa nel bilancio globale.

Il metodo viene applicato in maniera ricorsiva per eseguire le moltiplicazioni dei numeri di lunghezza inferiore, dividendo ogni volta in due le singole parti, fino a raggiungere una soglia di lunghezza minima dei fattori sotto la quale non è opportuno scendere; infatti, all'aumentare del numero dei passi del metodo, il risparmio in termini di moltiplicazioni viene bilanciato o addirittura superato dall'aumentato costo in termini di addizioni, sottrazioni e *shift* (le moltiplicazioni per le potenze della base B, che essenzialmente si riducono a spostamenti delle cifre verso sinistra). Per questo viene stabilito a priori un limite inferiore raggiunto il quale le moltiplicazioni vengono effettuate nella maniera tradizionale.

Si analizza ora il costo computazionale dell'algoritmo di Karatsuba. Se $k(n)$ è il tempo richiesto per moltiplicare numeri di n bit, abbiamo, per qualche costante c:

$$k(2n) \leq 3k(n) + cn, \qquad (1.3)$$

dal momento che il membro destro della (1.2) usa solo 3 moltiplicazioni più alcune addizioni e *shift*. Dalla (1.3) si ottiene, per induzione:

$$k(2^n) \leq c(3^n - 2^n), \quad k \geq 1,$$

pertanto abbiamo:

$$k(n) \leq k(2^{\lceil \log_2 n \rceil}) \leq c(3^{\lceil \log_2 n \rceil} - 2^{\lceil \log_2 n \rceil}) < 3c \cdot 3^{\log_2 n} = 3cn^{\log_2 3},$$

dove $\lceil x \rceil = \min_{z \in \mathbb{Z}} \{z \geq x\}$.

L'ultima relazione mostra che si può ridurre l'ordine di grandezza del costo computazionale di una moltiplicazione di numeri di n bit da n^2 a $n^{\log_2 3} \approx n^{1.585}$, quindi il metodo di Karatsuba è più veloce di quello tradizionale per n sufficientemente grande [8].

Un caso particolare dell'algoritmo si ha quando $a = b$, ossia quando si calcola il quadrato di un numero. In questo caso, il passo ricorsivo diventa:

$$a^2 = (1 + T)a_0^2 - T(a_1 - a_0)^2 + (T + T^2)a_1^2.$$

Esempio Vogliamo calcolare $225 \times 142 = 31950$. Supponiamo di lavorare in base 2: l'operazione diventa $11100001 \times 10001110 = 111110011001110$.

Col metodo tradizionale l'operazione ha il seguente svolgimento:

$$\begin{array}{rl} 11100001 & \times \\ 10001110 & = \\ \hline 00000000 & \\ 11100001 & \\ 11100001 & \\ 11100001 & \\ 00000000 & \\ 00000000 & \\ 00000000 & \\ 11100001 & \\ \hline\hline 111110011001110 & \end{array}$$

impiegando quindi 64 moltiplicazioni e 49 addizioni (oltre ai riporti).

Col metodo di Karatsuba, l'operazione si svolge così:

$$a = 11100001$$

$$b = 10001110$$

In questo caso $n = 8$, quindi $T = 2^4 = 16$. Spezziamo i fattori:

$$a_0 = 0001 \quad a_1 = 1110$$

$$b_0 = 1110 \quad b_1 = 1000$$

Effettuiamo quindi il primo passo dell'algoritmo:

$$\begin{aligned} ab &= (1+T)0001 \times 1110 + T(1110 - 0001)(1000 - 1110) + \\ &\quad + (T + T^2)1110 \times 1000 \\ &= (1+T)0001 \times 1110 - T(1101 \times 0110) + (T + T^2)1110 \times 1000 \end{aligned}$$

Ciascuna delle 3 moltiplicazioni può essere ulteriormente suddivisa in 3 moltiplicazioni di numeri di 2 cifre.

1.1.2 La trasformata discreta di Fourier

Il metodo di Karatsuba è competitivo per numeri dell'ordine delle centinaia di cifre; per numeri più grandi, un metodo più efficace è quello della trasformata discreta di Fourier (DFT, Discrete Fourier Transform), di cui qui è descritto l'algoritmo di calcolo FFT (Fast Fourier Transform) [9].

Definizione 2 *Sia dato un vettore $a = (a_0, a_1, \ldots, a_{n-1})$, di lunghezza n e sia $\omega = \exp\left(\frac{2i\pi}{n}\right)$ una radice n-esima dell'unità, dove i è l'unità immaginaria, ossia tale che $i^2 = -1$.*

1. *Si definisce* trasformata discreta di Fourier (DFT) *di a, il vettore:*

$$\mathcal{F}_n(a) = (a_0^*, a_1^*, \ldots, a_{n-1}^*), \quad a_k^* = \frac{1}{n}\sum_{j=0}^{n-1} a_j \omega^{-jk}, \quad 0 \leq k \leq n-1.$$

2. *Si definisce* trasformata discreta inversa di Fourier (IDFT) *di* a*, il vettore:*

$$\bar{\mathcal{F}}_n(a) = (a_0^*, a_1^*, \ldots, a_{n-1}^*), \quad a_k^* = \sum_{j=0}^{n-1} a_j \omega^{jk}, \quad 0 \leq k \leq n-1.$$

Si verifica facilmente che $\mathcal{F}_n(\bar{\mathcal{F}}_n(a)) = \bar{\mathcal{F}}_n(\mathcal{F}_n(a)) = a$, il che giustifica il termine di trasformata inversa.

La FFT è un algoritmo per calcolare la trasformata discreta inversa di Fourier di un vettore di lunghezza n in un tempo $O(n \log n)$ invece di $O(n^2)$, come invece si avrebbe col metodo classico. Nel seguito supporremo che n sia una potenza di 2.

Con le notazioni precedenti, il principio alla base della FFT è di scrivere:

$$a_k^* = \sum_{j=0}^{2n-1} a_j \omega^{jk} = \sum_{j=0}^{n-1} a_{2j} (\omega^2)^{jk} + \omega^k \sum_{j=0}^{n-1} a_{2j+1} (\omega^2)^{jk}.$$

Quindi, per calcolare i coefficienti a_k^* di $\bar{\mathcal{F}}_{2n}(a)$ tramite la FFT, si compiono i seguenti passi:

1. definiamo due vettori di lunghezza n:

$$a_P = (a_0, a_2, \ldots, a_{2n-2}), \quad a_D = (a_1, a_3, \ldots, a_{2n-1});$$

2. ne calcoliamo le trasformate inverse di Fourier:

$$p = \bar{\mathcal{F}}_n(a_P) = (p_0, p_1, \ldots, p_{n-1}), \quad d = \bar{\mathcal{F}}_n(a_D) = (d_0, d_1, \ldots, d_{n-1});$$

3. calcoliamo la trasformata inversa di Fourier $a^* = (a_0^*, a_1^*, \ldots, a_{2n-1}^*) = \bar{\mathcal{F}}_{2n}(A)$ tramite le formule:

$$a_k^* = p_k + \omega^k d_k, \quad a_{n+k}^* = p_k - \omega^k d_k, \qquad 0 \leq k \leq n-1.$$

Perciò il costo $T(2n)$ del calcolo di $\bar{\mathcal{F}}_{2n}(a)$ con il metodo FFT soddisfa la relazione ricorsiva $T(2n) = 2T(n) + O(n)$. Quando n è una potenza di 2, il processo può essere iterato fino a raggiungere il limite ottimale $T(n) = O(n \log n)$. Ovviamente l'algoritmo per effettuare la trasformata diretta è simile a questo.

Mostriamo adesso in che modo il problema della moltiplicazione di due numeri si può ridurre al calcolo della IDFT di un vettore mediante la FFT.

Siano a e b due numeri positivi lunghi al più $n-1$ cifre in base B e siano $a(z)$ e $b(z)$ i polinomi ad essi associati:

$$a(z) = \sum_{j=0}^{n-1} a_j z^j, \quad b(z) = \sum_{j=0}^{n-1} b_j z^j,$$

dimodoché

$$a = a(B), \quad b = b(B).$$

Un polinomio di grado minore di m è univocamente determinato dai valori che esso assume in m punti distinti (ad esempio, tramite interpolazione di Lagrange). Pertanto, per ottenere i coefficienti del polinomio $c(z)$, basta calcolarne i valori $c(w_k)$ in $2n$ punti distinti, ossia calcolare $a(w_k)$ e $b(w_k)$. L'idea della FFT consiste nello scegliere come w_k le radici $2n$-esime complesse dell'unità:

$$w_k = \exp\left(\frac{2ik\pi}{2n}\right) = \omega^k, \quad \omega = \exp\left(\frac{2i\pi}{2n}\right), \quad 0 \leq k \leq 2n-1.$$

Con questa scelta dei w_k, abbiamo due proprietà:

1. gli insiemi di valori $(a(w_0), \ldots, a(w_{2n-1}))$ e $(b(w_0), \ldots, b(w_{2n-1}))$ possono essere calcolati in un tempo $O(n \log n)$;

2. dai valori $a(w_k)$, $b(w_k)$, $k = 1, \ldots, 2n-1$ si può ricavare il polinomio $c(z)$ in un tempo $O(n \log n)$.

L'ultimo punto discende dal fatto che il k-esimo coefficiente c_k di $c(z)$ soddisfa la seguente proprietà:

$$c_k = \frac{1}{2n} t(\bar{w_k}), \quad t(z) = \sum_{j=0}^{2n-1} c(w_j) z^j.$$

e dal fatto che vale $c(w_k) = a(w_k)b(w_k)$ per $1 \leq k \leq 2n-1$.

In pratica, si riconduce la moltiplicazione di due numeri al prodotto componente per componente dei vettori derivati dalla convoluzione dei due polinomi rappresentanti i numeri in questione. Tale operazione, come vedremo, ha un costo globale di $O(n \log^3 n)$, inferiore quindi sia a quello dell'algoritmo banale della moltiplicazione, sia a quello del metodo di Karatsuba visto sopra.

Mostriamo ora l'algoritmo di moltiplicazione tramite FFT: sia n una potenza di 2, a e b due numeri con non più di n cifre e scriviamoli come polinomi nella base B:

$$a = \sum_{j=0}^{n-1} a_j B^j, \quad b = \sum_{j=0}^{n-1} b_j B^j$$

Per calcolare $c = ab$ in un tempo $O(n \log^3 n)$, si compiono i seguenti passi:

1. Tramite FFT, si calcola la trasformata inversa di Fourier a^* di lunghezza $2n$ del vettore (a_j):

$$a^* = (a_0^*, a_1^*, \ldots, a_{2n-1}^*) = \bar{\mathcal{F}}_{2n}(a_0, a_1, \ldots, a_{n-1}, 0, \ldots, 0);$$

2. facciamo lo stesso per calcolare b^*:

$$b^* = (b_0^*, b_1^*, \ldots, b_{2n-1}^*) = \bar{\mathcal{F}}_{2n}(b_0, b_1, \ldots, b_{n-1}, 0, \ldots, 0);$$

3. calcoliamo c^* moltiplicando termine a termine a^* e b^*:

$$c^* = (c_0^*, c_1^*, \ldots, c_{2n-1}^*), \quad c_k^* = a_k^* b_k^*. \quad 0 \leq k \leq 2n-1;$$

4. calcoliamo la trasformata diretta $\hat{c}$ di c^*:

$$\hat{c} = (\hat{c}_0, \hat{c}_1, \ldots, \hat{c}_{2n-1}) = \mathcal{F}_{2n}(c^*);$$

5. infine, dopo aver applicato le formule (1.1), otteniamo le cifre c_j, $j = 0$, $\ldots$, $2n-1$, tali che il numero

$$c = \sum_{j=0}^{2n-1} c_j B^j$$

è uguale al prodotto di a e b.

Notare che a^* e b^* sono le trasformate inverse di Fourier delle successioni ottenute accodando n cifre 0 rispettivamente ad a e b.

L'algoritmo perciò consiste nel calcolo di due trasformate inverse di Fourier di lunghezza $2n$ di numeri di una sola cifra, $2n$ moltiplicazioni di dati elementari che possono essere rappresentati con $\log 2n$ cifre significative e una trasformata diretta di lunghezza $2n$.

Se per calcolare le operazioni fra numeri rappresentabili con $\log_2 2n$ cifre viene utilizzato l'algoritmo tradizionale che richiede $O((\log_2 2n)^2)$ operazioni binarie, il costo globale del metodo FFT per il calcolo del prodotto di interi diventa $O(n \log n \log^2 n) = O(n \log^3 n)$. È possibile però riapplicare le stesse tecniche FFT per il calcolo dei prodotti fra numeri di $\log_2 2n$ cifre ottenendo il costo $O(n \log^2 n (\log \log n)^3)$.

Per quanto riguarda il calcolo del quadrato di un numero in alta precisione, è possibile risparmiare una trasformata discreta inversa di Fourier ad ogni passo ricorsivo. Ciò non modifica la stima asintotica del costo, ma riduce la costante moltiplicativa.

L'algoritmo di Schönhage-Strassen si basa sulle idee sopra esposte e, mediante l'uso della DFT su campi finiti, raggiunge il costo di $O(n \log n \log \log n)$ operazioni fra bit.

1.2 Divisione, radice quadrata e radice cubica

La moltiplicazione non è l'unica operazione dispendiosa che appare nel calcolo di funzioni elementari. Anche la divisione e il calcolo di radici sono operazioni costose che si cerca di minimizzare, sia nel numero che nel costo computazionale. A questo scopo, si possono utilizzare dei metodi di iterazione funzionale.

Sia data un'equazione di tipo $f(x) = 0$ che abbia una radice α all'interno di un intervallo $[a, b]$: un metodo di iterazione funzionale consiste nel costruire una successione della forma $x_{n+1} = g(x_n)$ che converga ad α, che quindi è un punto fisso di $g(x)$, ossia una soluzione dell'equazione

$$g(x) = x.$$

Si osservi che la funzione $f(x) = x^{-1} - a$ ha come radice il valore $x = 1/a$, mentre la funzione $f(x) = x^2 - a$ ha come radice il valore $x = \sqrt{a}$.

Riveste un'importanza notevole per il buon funzionamento del metodo la scelta di una adeguata funzione di iterazione $g(x)$. Uno dei modi per formare $g(x)$ è quello di scegliere una funzione $h(x)$ in modo che l'equazione

$$x = x - \frac{f(x)}{h(x)} \tag{1.4}$$

abbia le stesse soluzioni di quella originale in un opportuno sottointervallo di $[a, b]$ che contenga α.

Definizione 3 *Sia $\{x_n\}$ una successione convergente ad α e sia $x_n \neq \alpha$ per ogni n. Se esiste un numero reale $p \geq 1$ tale che*

$$\lim_{n\to\infty} \frac{|x_{n+1} - \alpha|}{|x_n - \alpha|^p} = \gamma, \ \textit{con} \begin{cases} 0 < \gamma \leq 1 & \textit{se } p = 1, \\ \gamma > 0 & \textit{se } p > 1, \end{cases}$$

si dice che la successione ha ordine di convergenza p. *La costante* γ *è detta* fattore di convergenza.

Se $p = 1$ *e* $0 < \gamma < 1$, *si dice anche che la convergenza è* lineare,

se $p = 1$ *e* $\gamma = 1$, *si dice anche che la convergenza è* sublineare,

se $p > 1$, *si dice anche che la convergenza è* superlineare *[10].*

1.2.1 Il metodo di Newton

Se $f(x)$ è una funzione derivabile, ponendo nella (1.4) $h(x) = f'(x)$, si ha:

$$x = g(x) = x - \frac{f(x)}{f'(x)}.$$

Il corrispondente metodo è detto *metodo delle tangenti* o *di Newton*:

$$x_{j+1} = x_j - \frac{f(x_j)}{f'(x_j)}, \quad f'(x_j) \neq 0, \quad j \geq 0.$$

Si dimostra che, se $f(x)$ è una delle funzioni di nostro interesse (reciproco, radice n-esima), il metodo delle tangenti ha ordine di convergenza 2 [10].

1.2.2 Divisione

L'algoritmo banale per effettuare la divisione è estremamente inefficiente per approssimare il risultato con elevata precisione. Molto meglio, invece, è calcolare $\frac{a}{b}$ come $a \cdot \frac{1}{b}$. L'inverso di b è calcolato, a partire da una prima approssimazione x_0, effettuando l'iterazione

$$x_{k+1} = x_k + x_k(1 - bx_k),$$

che si ottiene applicando il metodo di Newton alla funzione $f(x) = x^{-1} - b$, finché non si raggiunge la precisione richiesta. Infatti, la successione $\{x_n\}$ converge in modo quadratico (2° ordine), ossia il numero di cifre esatte si raddoppia ad ogni iterazione: se $x_k = \frac{1}{b}(1 + \varepsilon)$, allora

$$\begin{aligned} x_{k+1} &= \frac{1}{b}(1+\varepsilon) + \frac{1}{b}(1+\varepsilon)(1 - b \cdot \frac{1}{b}(1+\varepsilon)) \\ &= \frac{1}{b}(1-\varepsilon^2), \end{aligned}$$

ossia l'errore relativo viene elevato al quadrato ad ogni passo.

Inoltre, ogni passo richiede solo calcoli fatti con il doppio della precisione iniziale. Ancora meglio, la moltiplicazione $x_k \cdot (\dots)$ può essere effettuata con metà della precisione richiesta, in quanto calcola solo le cifre "di correzione" (che alterano solo la metà meno significativa delle cifre). Il costo computazionale totale in termini di moltiplicazioni è quindi:

$$1.5 \cdot \sum_{n=0}^{+\infty} \left(\frac{1}{2}\right)^n$$

ossia l'equivalente di meno di 3 moltiplicazioni in precisione completa. Contando anche l'ultima moltiplicazione, una divisione costa come 4 moltiplicazioni. Un altro pregio di questo algoritmo è che è autocorrettore, ossia eventuali errori introdotti in un generico passo vengono corretti nei passi successivi.

1.2.3 Radice quadrata

Per calcolare la radice quadrata di un numero reale d, se si tenta di applicare l'algoritmo di Newton alla funzione ovvia, $f(x) = x^2 - d = 0$, si ottiene l'iterazione

$$x_{k+1} = \frac{1}{2}(x_k + \frac{d}{x_k}).$$

Come si vede, in questa iterazione compaiono delle divisioni ad elevata precisione, che sono quattro volte più dispendiose di una moltiplicazione. Si

preferisce allora utilizzare l'iterazione

$$x_{k+1} = x_k + x_k \frac{(1 - dx_k^2)}{2}$$

che converge a $\frac{1}{\sqrt{d}}$ e che si ottiene applicando il metodo di Newton alla funzione $f(x) = x^{-2} - d$; una moltiplicazione finale per d consente di ottenere $\sqrt{d}$.

Anche in questo caso la convergenza è del secondo ordine. Con considerazioni simili alle precedenti (supponendo che il costo dell'elevamento al quadrato sia pari a quello di una moltiplicazione) otteniamo un costo computazionale di 4 moltiplicazioni per $\frac{1}{\sqrt{d}}$ e 5 per $\sqrt{d}$.

1.2.4 Radice cubica

Per il calcolo della radice cubica di un numero reale d, si possono fare considerazioni analoghe alle precedenti: se proviamo ad applicare il metodo di Newton alla funzione banale $f(x) = x^3 - d = 0$, si ottiene l'iterazione:

$$x_{k+1} = \frac{1}{3}(2x_k + \frac{d}{x_k^2})$$

che contiene anch'essa delle divisioni ad alta precisione. In questo caso è conveniente sfruttare l'identità $d^{1/3} = d(d^{-2/3})$, calcolare $d^{-2/3}$ tramite l'iterazione

$$x_{k+1} = x_k + x_k \frac{(1 - d^2 x_k^3)}{3},$$

che si ottiene applicando il metodo di Newton alla funzione $f(x) = x^{-3} - d$, e moltiplicare alla fine per d per ottenere $\sqrt[3]{d}$.

Anche per il calcolo della radice cubica si possono fare le stesse considerazioni dei due casi suddetti, riguardo alla velocità di convergenza e al costo computazionale.

1.3 Altri strumenti utili

1.3.1 Il metodo di esponenziazione binaria

Il metodo di esponenziazione binaria è un metodo rapido per elevare un numero reale x ad una potenza intera n.

1. Si scrive lo sviluppo binario di n; ciò può essere fatto tramite divisioni successive per 2 e scrivendo i resti in ordine inverso:
$$n = \sum_{j=0}^{k} b_j 2^j, \quad b_j \in \{0, 1\};$$
2. si calcolano le potenze di x elevato alle potenze di 2 fino alla k-esima tramite elevamenti successivi al quadrato:
$$x^{2^{j+1}} = (x^{2^j})^2, \quad 0 \leq j \leq k-1;$$
3. si calcola x^n moltiplicando fra loro le varie potenze di x ricavate al punto precedente, scegliendo quelle corrispondenti alle cifre 1 nello sviluppo binario di n:
$$x^n = \prod_{\substack{j=0 \\ b_j=1}}^{k} x^{2^j}.$$

Questo metodo può essere ugualmente utilizzato anche per calcolare la radice n-esima di un numero reale per n qualsiasi:

1. si considera lo sviluppo binario di $\frac{1}{n}$:
$$\frac{1}{n} = \sum_{j=0}^{+\infty} \frac{b_j}{2^j}, \quad b_j \in \{0, 1\};$$
si assume che lo sviluppo sia finito, cioè:
$$\frac{1}{n} = \sum_{j=0}^{k} \frac{b_j}{2^j},$$

altrimenti si dovrà ricorrere ad un'approssimazione (ad esempio, troncando lo sviluppo ad un certo k fissato);

2. si calcolano le radici 2^j-esime di x fino alla k-esima potenza tramite radici quadrate successive (calcolate col metodo visto in precedenza):

$$\sqrt[2^{j+1}]{x} = (\sqrt[2^j]{x})^{1/2};$$

3. si calcola $\sqrt[n]{x}$ moltiplicando fra loro le radici corrispondenti alle cifre "1" dello sviluppo binario di $\frac{1}{n}$:

$$\sqrt[n]{x} = \prod_{\substack{j=0 \\ b_j=1}}^{k} x^{\frac{1}{2^j}}.$$

La complessità di questa tecnica dipende dal valore di k. Inoltre, se lo sviluppo non è finito, per avere un'accuratezza di d cifre dovrà essere $2^{-k+1} < 2^{-d}$, per cui $k > d + 1$.

1.3.2 La formula di Stirling

La seguente formula, dovuta a J. Stirling (1730), permette di approssimare con grande precisione il fattoriale di un numero naturale n:

$$n! \approx \sqrt{2\pi n} n^n e^{-n}, \quad n \in \mathbb{N}. \tag{1.5}$$

Si dimostra [10, 11] che l'errore relativo commesso nell'approssimare $n!$ con la (1.5) è minore di $e^{\frac{1}{12n}}$.

1.3.3 Il metodo di Ruffini-Horner

Si consideri il polinomio di grado n

$$p(x) = \sum_{i=0}^{n} a_i x^i.$$

Il calcolo di $p(x)$ per un valore assegnato di x, può essere effettuato calcolando separatamente le potenze x^i mediante il seguente algoritmo

$$\begin{aligned} p_0 = a_0, \quad y_0 = 1, \\ y_i = y_{i-1}x, \quad p_i = a_i y_i + p_{i-1}, \quad i = 1, 2, \ldots, n, \\ p(x) = p_n, \end{aligned}$$

che richiede $2n - 1$ moltiplicazioni ed n addizioni. Poiché il polinomio $p(x)$ può anche essere rappresentato nel modo seguente

$$p(x) = (\ldots((a_n x + a_{n-1})x + a_{n-2})x + \cdots + a_1)x + a_0,$$

si può ottenere un altro algoritmo per il calcolo di $p(x)$, detto *metodo di Ruffini-Horner* o della *divisione sintetica*:

$$\begin{aligned} p_0 = a_n, \\ p_i = p_{i-1}x + a_{n-i}, \quad i = 1, 2, \ldots, n, \\ p(x) = p_n, \end{aligned}$$

che impiega n moltiplicazioni ed n addizioni, cioè il numero di moltiplicazioni è dimezzato rispetto al metodo precedente. È stato dimostrato [12] che il metodo di Horner è ottimale nel caso in cui il polinomio venga individuato mediante i suoi coefficienti [10].

1.3.4 Serie ipergeometriche

Definizione 4 *Una* serie ipergeometrica *è una serie di potenze in cui il rapporto fra due coefficienti successivi è una funzione razionale dell'indice k:*

$$ {}_pF_q(a_1, \ldots, a_p; b_1, \ldots, b_q; x) = \sum_{k=0}^{+\infty} c_k x^k, \tag{1.6}$$

$$ \frac{c_{k+1}}{c_k} = \frac{(k + a_1)(k + a_2)\ldots(k + a_p)}{(k + b_1)(k + b_2)\ldots(k + b_q)}. \tag{1.7}$$

Capitolo 2

Calcolo della funzione esponenziale

Io stimo più il trovar un vero benché di cosa leggiera ché il disputar lungamente delle massime questioni senza conseguir verità nissuna.

Galileo Galilei

Fra le funzioni elementari, un ruolo fondamentale è rivestito dalle funzioni esponenziali, in particolare dalla funzione $f(x) = e^x$, che ha per base il numero di Nepero. Questa importanza è data dal fatto che a partire dall'esponenziale è possibile calcolare rapidamente altre funzioni elementari semplici, quali ad esempio il logaritmo naturale e le funzioni trigonometriche.

2.1 Metodi derivanti dalla serie di Taylor

Nel seguito, indicheremo con $t_n(f)$ il tempo necessario per calcolare la funzione f in precisione n, espresso in termini di numero di operazioni fra bit; in particolare, con $t_n(M)$ indicheremo il tempo necessario per effettuare una moltiplicazione di due numeri con n cifre (ossia, coi metodi visti nel capitolo precedente, $t_n(M) = n \log n \log\log n$).

2.1.1 Introduzione

Il metodo più diffuso per il calcolo della funzione esponenziale è quello basato sullo sviluppo in serie di Taylor:

$$\exp x = \sum_{n=0}^{+\infty} \frac{x^n}{n!}. \tag{2.1}$$

Questo metodo, però, ha il difetto di avere una convergenza molto lenta: si dimostra infatti che servono $\frac{c_1 n}{\log n}$ termini per avere una convergenza in un tempo $t_n(\exp) \leq c_2 t_n(M) \frac{n}{\log n}$, dove c_1 e c_2 sono costanti indipendenti da n.

Un primo raffinamento del metodo può essere ottenuto riducendo l'argomento tramite varie identità prima di procedere al calcolo della serie e poi invertendo la riduzione alla fine per ottenere il risultato finale. La più famosa ed una delle più utili a questo scopo è:

$$\exp x = \left(\exp \frac{x}{2^k}\right)^{2^k}. \tag{2.2}$$

Essa può essere usata in questo modo: calcoliamo $y = x/2^k$, che richiede soltanto divisioni per 2 riconducibili a *shift*; poi calcoliamo $\exp y$ sommando gli addendi della (2.1) fino a raggiungere la precisione richiesta; infine, effettuiamo k elevamenti al quadrato per ottenere $\exp x$.

In tal caso si ha $t_n(\exp) \leq c_3 \sqrt{n} t_n(M)$. Infatti, scegliamo $\lambda = 2^q$, dove $q = \lfloor \sqrt{n} \rfloor$; inoltre, sia $[a, b]$ il dominio in cui varia x e sia $c = \max(|a|, |b|)$;

allora:

$$\left|\frac{(x/\lambda)^r}{r!}\right| \leq 2^{-qr},$$

se r è abbastanza grande, in modo che $c^r \leq r!$. È sufficiente allora prendere $r = \lceil n/q \rceil \approx \sqrt{n}$ termini della serie di potenze di $e^{x/\lambda}$ per avere un errore assoluto dell'ordine di 2^{-n}. A questo punto, con q elevamenti al quadrato, si ottiene e^x a partire da $e^{x/\lambda}$ [13].

2.1.2 Un miglioramento del metodo di Taylor

Per precisioni di poche centinaia di cifre, si possono avere sensibili miglioramenti del tempo di convergenza utilizzando una versione migliorata del metodo di Taylor, descritta da Smith [6].

Poiché i termini della serie di Taylor (2.1) sono tutti della forma $x^k/k!$, si può passare da un termine al successivo tramite una divisione per un intero e una moltiplicazione per x, che richiede un tempo di $O(t_n(M))$. Poiché le addizioni e le divisioni per interi sono tutte operazioni $O(n)$, è importante ridurre il numero delle moltiplicazioni, che sono più costose. La somma

$$\exp x \approx 1 + x + \frac{x^2}{2} + \frac{x^3}{3!} + \cdots + \frac{x^t}{t!}$$

richiede $t-1$ moltiplicazioni, $t-1$ divisioni per interi e $t-1$ addizioni.

Gli addendi possono essere riordinati in questo modo:

$$\begin{array}{rl}
1 & +x^j/j! + x^{2j}/(2j)! + \ldots \\
+x[1 & +x^j/(j+1)! + \ldots \\
+x^2[1/2! & +x^j/(j+2)! + \ldots \\
+x^3[1/3! & +x^j/(j+3)! + \ldots \\
\vdots & \vdots \\
+x^{j-1}[1/(j-1)! & +x^j/(2j-1)! + \ldots .
\end{array}$$

Definiamo, per semplicità:

$$S_k = \sum_{r=0}^{+\infty} \frac{x^{rj}}{(rj+k)!}, \quad 0 \le k \le j-1.$$

Per poter aggiungere il successivo addendo a ciascuna delle S_k, servono una moltiplicazione per ottenere la successiva potenza di x^j, j divisioni per interi e j addizioni. Il termine originale x^j può essere ottenuto in $O(\log j)$ moltiplicazioni usando il metodo di esponenziazione binaria descritto in precedenza. Indi, il polinomio $S_{j-1}x^{j-1}+\cdots+S_1x+S_0$ viene calcolato col metodo di Ruffini-Horner che richiede $(j-1)$ moltiplicazioni e $(j-1)$ addizioni. Per alte precisioni, questo vuol dire che il numero di moltiplicazioni necessarie per calcolare $\exp x$ è circa $t/j + j$ e il numero di operazioni dell'ordine di $O(n)$ è lo stesso del calcolo della serie di Taylor.

Per dare una stima del tempo richiesto da questo algoritmo, supponiamo che l'argomento sia dell'ordine di grandezza di 1, che venga usata un'aritmetica in base b con n cifre di precisione e che vengano effettuati k dimezzamenti prima di calcolare le j somme. Supponiamo inoltre che l'argomento di partenza x giaccia in un qualche intervallo limitato fissato e che il numero t di termini necessari per il calcolo della serie di Taylor sia un intero in precisione singola. Il valore di t è determinato dall'equazione:

$$\frac{(2^{-k})^t}{t!} = b^{-n}.$$

Usando l'approssimazione di Stirling (1.5) per $t!$, otteniamo un'approssimazione del numero di termini della serie necessari:

$$t \approx \frac{n \log b}{\log n + k \log 2}.$$

Considerando anche i k elevamenti al quadrato necessari a invertire la riduzione dell'argomento, il numero totale di moltiplicazioni è stimabile in:

$$W \approx \frac{n \log b}{j(\log n + k \log 2)} + j + k.$$

Calcolando i valori di j e k che minimizzino W, troviamo:

$$j = n^{1/3}(\log b/\log 2)^{1/3},$$
$$k = n^{1/3}(\log b/\log 2)^{1/3} - \log n/\log 2.$$

Se prendiamo come valori di j e k gli interi più vicini a questi valori, otteniamo un algoritmo con $O(n^{1/3})$ moltiplicazioni; se ne conclude che $\exp x$ può essere calcolato con n cifre di precisione in un tempo $O(n^{1/3}t_n(M))$.

Se si usa un algoritmo di moltiplicazione di costo inferiore a $O(n^2)$, allora il tempo richiesto da tutte le $O(n)$ operazioni diventa sufficientemente alto da modificare i valori ottimi di j e k. Poiché vi sono $O(t)$ addizioni e divisioni per interi, con $t = O(n/k)$, il tempo di calcolo di $\exp x$ può essere stimato in:

$$T \approx \left(\frac{n\log b}{j(\log n + k\log 2)} + j + k\right) t_n(M) + \frac{2n^2}{k}.$$

Il calcolo dei valori di j e k che minimizzino T dà risultati diversi dai precedenti quando $t_n(M) = o(n^{4/3})$. In questo caso i valori ottimi sono $j = O(\sqrt[4]{t_n(M)})$ e $k = O(n/\sqrt{t_n(M)})$ e l'algoritmo viene eseguito in un tempo $O(n\sqrt{t_n(M)})$. Quindi, se si usa un algoritmo moltiplicativo molto veloce, sono necessarie meno addizioni e viene effettuata una maggior riduzione dell'argomento.

2.2 Un metodo basato sulle approssimanti di Padé

Sempre per basse precisioni, è possibile migliorare ulteriormente l'algoritmo precedente utilizzando, al posto della serie di Taylor (o, in questo caso, di Maclaurin), le approssimanti di Padé. Il metodo presentato di seguito è descritto in [7].

2.2.1 Definizioni

Definizione 5 *La funzione razionale*

$$R_{m,n}(x) = \frac{p_m(x)}{q_n(x)},$$

dove $p_m(x)$ *e* $q_n(x)$ *sono due polinomi di grado rispettivamente minore o uguale a* m *e a* n *e* $q_n(0) \neq 0$*, è detta* approssimante di Padé *di ordine* $m+n$ *della funzione* $f(x)$ *se*

$$R^{(k)}_{m,n}(0) = f^{(k)}(0), \quad k = 0, \ldots, m+n,$$

cioè se gli sviluppi di Maclaurin di $f(x)$ *e di* $R_{m,n}(x)$ *coincidono fino al termine* $(m+n+1)$*-esimo.*

In particolare, per $n = 0$, si ha:

$$R_{m,0}(x) = p_m(x) = \sum_{j=0}^{m} \alpha_j x^j,$$

cioè $R_{m,0}$ è il polinomio ottenuto troncando all'$(m+1)$-esimo termine la serie di Maclaurin di $f(x)$.

Nel caso della funzione esponenziale, dalla proprietà

$$e^x = \frac{1}{e^{-x}}$$

segue subito che

$$R_{m,n}(x) = \frac{1}{R_{n,m}(-x)}$$

e quindi per calcolare le approssimanti di Padé è necessario calcolare solo i coefficienti del polinomio a numeratore [10, 14].

2.2.2 Descrizione dell'algoritmo

Si inizia, come di consueto, con una riduzione dell'argomento dell'esponenziale. Supponiamo che $P = 10^m$. Se s è l'argomento dell'esponenziale, poniamo

$$e^s = 10^n \cdot 10^t = 10^n \cdot e^{t \log 10} = P^l \cdot 10^r \cdot e^y,$$

con

$$n = lm + r, \quad 0 \leq r \leq m; \quad y = t \log 10 \in (-\log 10, 0].$$

La y così ottenuta viene ulteriormente ridotta utilizzando k volte la formula (2.2), riconducendosi così al calcolo di e^x, con $x = 2^{-k}y$.

Sia ora $\frac{P_m(z)}{Q_n(z)}$ l'approssimante di Padé $[m/n]$ di e^z; inoltre, denotiamo con ${}_1F_1(a_1; b_1; x)$ la serie ipergeometrica definita in (1.6). Allora abbiamo che:

$$P_m(z) = {}_1F_1(-m, -(m+n); z) \quad Q_n(z) = {}_1F_1(-n, -(m+n); -z),$$

col resto scritto nella forma

$$e^z - \frac{P_m(z)}{Q_n(z)} = \frac{m!n!}{(m+n)!(m+n+1)!} z^{m+n+1} + O(z^{m+n+2}). \tag{2.3}$$

Prendiamo $m+n$ pari. Dalla (2.3) ricaviamo che, se $m+n$ rimane costante (e quindi la quantità totale di operazioni per calcolare $P_m(z)$ e $Q_n(z)$ rimane costante), l'errore è minimo quando $m = n$. In questo caso allora abbiamo

$$Q_m(z) = P_m(-z),$$

dove

$$P_m(z) = 1 + \frac{m}{2m}\frac{z}{1!} + \frac{m(m-1)}{2m(2m-1)}\frac{z^2}{2!} + \cdots + \frac{m!}{2m(2m-1)\ldots(m+1)}\frac{z^m}{m!}.$$

Pertanto se poniamo:

$$\alpha(z) = 1 + \frac{m(m-1)}{2m(2m-1)}\frac{z^2}{2!} + \frac{m(m-1)(m-2)(m-3)}{2m(2m-1)(2m-2)(2m-3)}\frac{z^4}{4!} + \ldots,$$

$$\beta(z) = \frac{m}{2m} + \frac{m(m-1)(m-2)}{2m(2m-1)(2m-2)}\frac{z^2}{3!} + \frac{m(m-1)\ldots(m-4)}{2m(2m-1)\ldots(2m-4)}\frac{z^4}{5!} + \ldots,$$

allora:

$$e^x \approx \frac{P_m(x)}{Q_m(x)} = \frac{\alpha(x) + x\beta(x)}{\alpha(x) - x\beta(x)}. \tag{2.4}$$

Il calcolo di $\alpha(x)$ e $\beta(x)$ richiede un approccio simile a quello dell'algoritmo precedente per la serie di Taylor (2.1) dell'esponenziale, ossia dividere $P_m(x)$ in j segmenti (scelto j pari):

$$\begin{aligned} P_m(z) = &\left[1 + \frac{m(m-1)\dots(m-j+1)}{2m(2m-1)\dots(2m-j+1)}\frac{z^j}{j!} + \frac{m\dots(m-2j+1)}{2m\dots(2m-2j+1)}\frac{z^{2j}}{(2j)!} + \dots\right] \\ &+ z\left[\frac{m}{2m} + \frac{m(m-1)\dots(m-j)}{2m(2m-1)\dots(2m-j)}\frac{z^j}{(j+1)!} + \dots\right] \\ &+ \dots \\ &+ z^{j-1}\left[\frac{m(m-1)\dots(m-j+2)}{2m(2m-1)\dots(2m-j+2)}\right. \\ &\left.+\frac{m(m-1)\dots(m-2j+2)}{2m(2m-1)\dots(2m-2j+2)}\frac{z^j}{(2j-1)!} + \dots\right]. \end{aligned} \tag{2.5}$$

Calcoliamo la somma per ogni segmento in modo da ottenere le somme parziali $S_0, S_1, \dots, S_{j-1}$ (definite in maniera analoga a quanto abbiamo fatto nel paragrafo precedente) e poi calcoliamo:

$$\alpha(z) = \sum_{i=0}^{(j-2)/2} (z^2)^i S_{2i}, \qquad \beta(z) = \sum_{i=1}^{j/2} (z^2)^{i-1} S_{2i-1}.$$

Confrontiamo (2.3) e (2.1). Poiché

$$\frac{(m!)^2}{(2m)!(2m+1)!} \ll \frac{1}{(2m+1)!},$$

per ottenere la stessa precisione occorrono molti meno termini per il calcolo di $P_m(x)/Q_m(x)$ rispetto a quelli richiesti dalla (2.1). Inoltre, in questo caso bisogna calcolare solo $P_m(x)$, il che riduce notevolmente il numero di operazioni richieste. Ovviamente, l'utilizzo della (2.4) incrementa il numero di divisioni; inoltre è richiesta un'ulteriore divisione per un intero per il calcolo delle somme in (2.5) rispetto a (2.1).

2.2.3 Complessità computazionale

Per un dato y, la quantità di operazioni necessarie al calcolo di e^y dipende da k, numero di dimezzamenti effettuati per applicare la (2.2), e da j.

Supponiamo di adottare un metodo per la moltiplicazione di numeri rappresentabili da N bit di costo $t_N(M) \leq c_m N^2$ e supponiamo che il costo di un'addizione o sottrazione sia $c_a N$ e quella di una moltiplicazione o divisione semplici (cioè in cui uno dei fattori o divisore sia un intero a precisione singola) sia $c_s N$. c_m, c_a e c_s sono costanti che possono essere ricavate sperimentalmente una volta fissati gli algoritmi per l'aritmetica in alta precisione.

La riduzione dell'argomento tramite la (2.2) richiede un numero di operazioni elementari pari a

$$W_k = c_m b_1 k N^2 + c_a b_2 k N,$$

dove, per il caso dell'esponenziale, $b_1 = 1$, $b_2 = 0$.

Il numero di operazioni per il calcolo di x^j (con j pari) usando il metodo di esponenziazione binaria descritto nel paragrafo 1.3.1 è pari a:

$$W_j = c_m b_3(j) \log j \cdot N^2$$

dove

$$1/\log 2 \leq b_3(j) \leq 2/\log 2.$$

Analizziamo ora la quantità di addizioni, moltiplicazioni semplici e divisioni necessarie per il calcolo di $P_m(x)$.

Sia $m = m(k)$ il minimo numero naturale che soddisfa la disuguaglianza

$$\frac{(m!)^2 |x|^{2m+1}}{(2m)!(2m+1)!} \leq P^{-N}$$

con

$$x = \eta^{-k} y, \quad \eta = 2.$$

Per la formula di Stirling (1.5), possiamo scegliere come $m(k)$ una approssimazione della radice dell'equazione

$$f(m) = m(\log m + a) - b = 0, \tag{2.6}$$

dove

$$a = -\log|x| - \log\left(\frac{e}{4}\right),$$
$$b = \frac{1}{2}[N \log P + \log|x| - \log 2].$$

Quando $m \geq 1$, si ha $f'(m) > 0$ e $f''(m) > 0$, quindi per ogni valore iniziale $m_0 \geq 1$, il metodo di iterazione di Newton

$$m_n = \frac{m_{n-1} + b}{\log m_{n-1} + a + 1}, \quad n \geq 1 \tag{2.7}$$

che risolve l'equazione (2.6) è convergente. Prendendo $m_0 = N$, possiamo determinare $m(k)$ iterando finché $|m_n - m_{n-1}| \leq 1/2$. Inoltre, abbiamo

$$m'(k) = -\frac{[m(k) + \frac{1}{2}] \log \eta}{\log[m(k)] + a + 1}. \tag{2.8}$$

Poniamo $P_m(x) = \sum_{i=0}^{m} a_i x^i$. Allora

$$a_i = [m!(2m - i)!]/[(m - i)!(2m)!i!].$$

Per la formula di Stirling (1.5) e per induzione, possiamo dedurne che

$$1/(4^i i!) \leq a_i \leq 1/(2^i i!). \tag{2.9}$$

Poniamo $a_i|x|^i = P^{-d(k,i)}$. Allora, per la (2.9) e la formula di Stirling (1.5), vale:

$$\begin{aligned} d(k,i) &\geq [\log(i!) + i \log(2/|x|)]/\log P \\ &\approx [i \log i - i + \frac{1}{2} \log i + \frac{1}{2} \log(2\pi) + i \log(2/|x|)]/\log P. \end{aligned}$$

Quindi la quantità necessaria di addizioni, moltiplicazioni e divisioni per costante ammonta a:

$$W_s = \sum_{i=1}^{m(k)} (c_a + 2c_s)[N - d(k,i)] \approx (c_a + 2c_s) \int_1^{m(k)} [N - d(k,i)]di.$$

La quantità di moltiplicazioni necessarie per il calcolo di $P_m(x)$ è:

$$\begin{aligned} W_m &= c_m j N^2 + \sum_{i=1}^{m(k)/j} c_m [N - d(k,ij)]^2 \\ &\approx c_m j N^2 + c_m \int_1^{m(k)/j} c_m [N - d(k,ij)]^2 di \\ &= c_m j N^2 + \frac{c_m}{j} \int_j^{m(k)} [N - d(k,s)]^2 ds. \end{aligned}$$

Sintetizzando ed applicando l'induzione, otteniamo la stima del numero totale di operazioni come:

$$\begin{aligned} W(k,j) &= W_k + W_j + W_s + W_m = c_m[b_1 k + j + b_3(j)\log j]N^2 + c_a b_2 kN \\ &\quad + (c_a + 2c_s) \int_1^{m(k)} [N - d(k,i)]di + \frac{c_m}{j} \int_j^{m(k)} [N - d(k,s)]^2 ds \\ &= W_1(k,j) + \frac{c_a + 2c_s}{\log P} W_2(k,j) + \frac{c_m}{j \log^2 P} W_3(k,j), \end{aligned}$$

dove

$$\begin{aligned} W_1(k,j) &= c_m[b_1 k + j + b_3(j)\log j]N^2 + c_a b_2 kN, \\ W_2(k,j) &= \int_1^{m(k)} (\alpha - s\log s - s\beta - \frac{1}{2}\log s)ds \\ &= -b_4(k,1) - \beta b_6(k,1) - \frac{1}{2} b_7(k,1) + \alpha b_8(k,1), \\ W_3(k,j) &= \int_j^{m(k)} (\alpha - s\log s - s\beta - \frac{1}{2}\log s)^2 ds = \sum_{i=0}^{8} c_i b_i(k,j), \end{aligned}$$

$$\alpha = N\log P - \frac{1}{2}\log(2\pi), \quad \beta = k\log\eta + \log(2/y) - 1,$$

$$b_i(k,j) = a_i(m(k)) - a_i(j), \quad i = 0, 1, \ldots 8,$$

e infine

$$a_0(s) = \frac{1}{3}(\log^2 s - \frac{2}{3}\log s + \frac{2}{9}), \quad a_1(s) = \frac{1}{3}s^3(\log s - \frac{1}{3}),$$
$$a_2(s) = \frac{1}{2}s^2(\log^2 s - \log s + \frac{1}{2}), \quad a_3(s) = \frac{1}{3}s^3, \quad a_4(s) = \frac{1}{2}s^2(\log s - \frac{1}{2}),$$
$$a_5(s) = s(\log s - 1)^2 + s, \quad a_6(s) = \frac{1}{2}s^2, \quad a_7(s) = s(\log s - 1), \quad a_8(s) = s,$$
$$c_0 = 1, \quad c_1 = 2\beta, \quad c_2 = 1, \quad c_3 = \beta^2, \quad c_4 = \beta - 2\alpha, \quad c_5 = \frac{1}{4},$$
$$c_6 = -2\alpha\beta, \quad c_7 = -\alpha, \quad c_8 = \alpha^2.$$

A questo punto si pone il problema della minimizzazione, ossia trovare due interi non negativi k^* e j^*, con j pari, tali che

$$W(k^*, j^*) = \min\{W(k, j) : k, \frac{1}{2}j \in \mathbb{N}\}.$$

Dal momento che $b_3(j)$ è definita solo sugli interi, non è possibile usare il metodo della derivata per trovare il minimo di W. Nel seguito cercheremo di determinare gli intervalli a cui appartengono k^* e j^* e proporremo diversi metodi per la soluzione approssimata. Tutte le operazioni effettuate in questa parte saranno eseguite in precisione di macchina.

Determinazione del limite superiore Allo scopo di determinare il limite superiore, trattiamo $W(k, j)$ con "valori di k e j rafforzati", ossia, sostituiamo $W(k, j)$ con:

$$\bar{W}(k, j) = c_m(b_1 k + j + \frac{\log j}{\log 2})N^2 + c_a b_2 kN + (c_a + 2c_s)m(k)N + \frac{c_m m(k)}{j}N^2.$$

Quindi, il punto minimo di $\bar{W}(k, j)$ soddisfa

$$m'(k) = -\frac{c_m b_1 N + c_a b_2}{c_a + 2c_s + c_m N/j}, \tag{2.10}$$

$$j = -\frac{1}{2\log 2} + \sqrt{(\frac{1}{2\log 2})^2 + m(k)}. \tag{2.11}$$

Determinazione del limite inferiore Allo stesso modo, per determinare il limite inferiore di j e k, dobbiamo trattare $W(j,k)$ con "valori di j e k indeboliti". A questo scopo, poniamo:

$$\bar{W}(k,j) = c_m(b_1k + j + 2\log j/\log 2)N^2 + c_a b_2 kN \\ + (c_a + 2c_s)m(k)[N - d(k,m(k))] + \frac{c_m m(k)}{j}[N - d(k,m(k))]^2.$$

In virtù della (2.6), la stima di $d(k,i)$ implica

$$d(k,m(k)) \approx \frac{1}{\log P}[b - m(k)\log 2 + \frac{1}{2}\log m(k) + \frac{1}{2}\log 2\pi].$$

Quindi,

$$N - d(k,m(k)) \approx \frac{1}{\log P}[N\log P - b + m(k)\log 2 - \frac{1}{2}\log m(k) - \frac{1}{2}\log 2\pi] \geq \frac{N}{2}.$$

Allora possiamo scegliere

$$\underline{W}(k.j) = c_m(b_1k + j + 2\log j/\log 2)N^2 + c_a b_2 kN \\ + \frac{1}{2}(c_a + 2c_s)m(k)N + c_m\frac{m_k}{4j}N^2.$$

Pertanto, il punto minimo di $\underline{W}(k.j)$ soddisfa

$$m'(k) = -\frac{2(c_m b_1 N + c_a b_2)}{c_a + 2c_s + c_m N/(2j)},$$
$$j = -\frac{1}{\log 2} + \sqrt{(\frac{1}{\log 2})^2 + \frac{m(k)}{4}}.$$

Algoritmo di ottimizzazione globale Per $j = 2, 4, 6, \ldots$ ($j \leq -\frac{1}{2\log 2} + \sqrt{(\frac{1}{2\log 2})^2 + m(0)} + \frac{1}{2}$), per $k = 0, 1, 2, \ldots$:

1. Calcolare $m(k)$ e $m'(k)$.

2. Verificare se

$$m'(k) \leq -2(c_m b_1 N + c_a b_2)/[c_a + 2c_s + c_m N/(2j)].$$

Se sì, passare al valore di k successivo, altrimenti passare al punto 3.

3. Verificare se

$$m'(k) \geq -(c_m b_1 N + c_a b_2)/(c_a + 2c_s + c_m N/j).$$

   Se sì, passare al valore di j successivo, altrimenti passare al punto 4.

4. Calcolare $W_1(k, j)$, $W_2(k, j)$, $W_3(k, j)$ e $W(k, j)$.

5. Confrontare e scegliere il valore minimo di $W(k, j)$ e memorizzare il punto minimo (k, j) e $m(k)$.

Algoritmo di ottimizzazione in direzione j Poiché gli errori di arrotondamento dovuti all'utilizzo della (2.2) si accumulano piuttosto in fretta, sotto determinate circostanze il valore di k non può essere scelto a piacimento. La cosiddetta ottimizzazione in direzione j consiste nello scegliere j tale che $W(k, j)$ raggiunga il minimo quando k è fissato. Ciò può essere implementato calcolando $W(k, 2)$, $W(k, 4)$, ..., $W(k, j)$ e scegliendo il minimo fra essi. Il valore massimo del numero pari j può essere scelto guardando l'estremo superiore (2.11).

Algoritmo di ottimizzazione in direzione k Se N è molto grande, il parametro j può assumere valori grandi; quindi, serve molto spazio per memorizzare tutti i valori richiesti dal calcolo. Questa situazione non è accettabile se si lavora con computer che hanno risorse di memoria e di spazio su disco limitate. La cosiddetta ottimizzazione in direzione k consiste nello scegliere k tale che $W(k, j)$ raggiunga il minimo quando j è fissato. Come nel punto precedente, Ciò può essere implementato per calcolo e confronto. Il valore massimo del numero k può essere scelto guardando l'estremo superiore (2.10).

Ottimizzazione locale Nel dominio fissato di (k, j) $(0 \leq k \leq K,\ 2 \leq j \leq J)$, calcoliamo $W(k, j)$ e troviamo (k^*, j^*) tali che

$$W(k^*, j^*) = \min\{W(k, j) : k, \frac{1}{2}j \in \mathbb{N}, 0 \leq k \leq K, 2 \leq j \leq J\}.$$

Schema di compromesso Se la velocità delle operazioni può essere ridotta di poco in modo da poter scegliere k e j più piccoli possibile, possiamo adottare il seguente schema di compromesso: dato un valore ammesso $T > 0$, che denoti di quanto si può aumentare la quantità di operazioni effettuate (ad esempio, $T = 0.02$ indica che si può aumentare il numero delle operazioni del 2%) scegliamo i minimi $k_1^* < k^*$ o $j_1^* < j^*$ tali che

$$W(k_1^*, j^*)/W(k^*, j^*) \leq 1 + T \tag{2.12}$$

oppure

$$W(k^*, j_1^*)/W(k^*, j^*) \leq 1 + T, \tag{2.13}$$

dove $W(k^*, j^*)$ è il minimo raggiunto da $W(k, j)$.

Supponiamo che N sia abbastanza grande. Trascurando i termini di ordine più alto, possiamo aumentare $W_3(k, j)$ fino a

$$\begin{aligned} W_3(k, j) &= \int_j^{m(k)} [\alpha - s(\log s + \beta)]^2 ds \\ &= \int_j^{m(k)} [\alpha^2 - 2\alpha s(\log s + \beta) + s^2(\log s + \beta)^2] ds \\ &\approx [\alpha^2 s - \alpha s^2(\log s + \beta) + \frac{1}{3}s^3(\log s + \beta)^2]\Big|_j^{m(k)}, \end{aligned}$$

omettiamo i termini relativi a j ed applichiamo la (2.2). Otteniamo così

$$\begin{aligned} W_3(k, j) &\approx m(k)[(N \log P)^2 - N \log P(\frac{1}{2}N \log P) + \frac{1}{3}(\frac{1}{2}N \log P)^2] \\ &= \frac{7}{12}m(k)(N \log P)^2. \end{aligned}$$

Otteniamo così la stima asintotica

$$W(k,j) \approx c_m(b_1+j)N^2 + \frac{7}{12}c_m\frac{m(k)}{j}N^2, \tag{2.14}$$

il cui punto di minimo soddisfa

$$j = [\frac{7}{12}m(k)]^{\frac{1}{2}} = [\phi m(k)] \quad (\phi = \frac{7}{12}), \tag{2.15}$$

$$m'(k) = -b_1 j\phi^{-1} = -b_1[m(k)/\phi]^{\frac{1}{2}}. \tag{2.16}$$

Per le (2.6)–(2.8), abbiamo:

$$m(k) \approx \frac{b}{\log N + a} \approx \frac{1}{2}\frac{N\log P}{logN + k\log\eta}, \tag{2.17}$$

$$m'(k) = -\frac{[m(k)+\frac{1}{2}]\log\eta}{[m(k)+b]/m(k)} \approx -\frac{m^2(k)\log\eta}{b}. \tag{2.18}$$

Combinando la (2.18) e la (2.16), otteniamo:

$$m(k) = \{b_1^2 b^2/[\phi\log^2\eta]\}^{\frac{1}{3}}. \tag{2.19}$$

Quindi, per la (2.15), segue che

$$j = \phi^{\frac{1}{3}}[b_1 b/\log\eta]^{\frac{1}{3}}. \tag{2.20}$$

Dalla (2.19) e dalla (2.17),

$$k = b_1^{-\frac{2}{3}}[\phi b/\log\eta]^{\frac{1}{3}} - \log N/\log\eta. \tag{2.21}$$

Sostituiamo le (2.19)–(2.21) nella (2.14), sostituiamo b col suo termine dominante $\frac{1}{2}N\log P$ e trascuriamo il termine di ordine inferiore $\log N/\log\eta$. Otteniamo così la stima asintotica della quantità minima di operazioni del nostro algoritmo:

$$W_f = \min\{W(k,j) : k, \frac{1}{2}j \in \mathbb{N}\} \approx 3c_m(\frac{1}{2}\phi b_1)^{\frac{1}{3}}[N\log P/\log\eta]^{\frac{1}{3}}N^2. \tag{2.22}$$

Confrontiamo questo risultato con quello ottenuto in precedenza tramite l'algoritmo di Smith:

$$W_{sf} = 3c_m[N \log P / \log 2]^{\frac{1}{3}} N^2; \tag{2.23}$$

il rapporto fra le due quantità è $W_{sf}/W_f = (\frac{1}{2}\phi b_1)^{-\frac{1}{3}} = (\frac{24}{7})^{\frac{1}{3}} \approx 1.50789$.

Ciò mostra che per una precisione che va dalle 20 alle 10000 cifre decimali, l'algoritmo di Smith richiede dal 30 al 50% di tempo di calcolo in più rispetto a questo.

2.3 AGM (Media Aritmetico-Geometrica)

Presentiamo adesso il metodo di calcolo basato sull'iterazione della media aritmetico-geometrica (AGM, *Arithmetic Geometric Mean*).

La trattazione di questo metodo riprende in massima parte l'esposizione dell'articolo di Brent [5].

2.3.1 Risultati sugli integrali ellittici

I risultati seguenti si trovano riassunti in [15].

Definizione 6 *1. Si definisce* integrale ellittico di prima specie *l'espressione*

$$F(\psi, \alpha) = \int_0^{\psi} (1 - \sin^2 \alpha \sin^2 \theta)^{-\frac{1}{2}} d\theta. \tag{2.24}$$

2. Si definisce integrale ellittico di seconda specie *l'espressione*

$$E(\psi, \alpha) = \int_0^{\psi} (1 - \sin^2 \alpha \sin^2 \theta)^{\frac{1}{2}} d\theta. \tag{2.25}$$

Per i nostri scopi, è sufficiente supporre che α e ψ appartengano all'intervallo $[0, \pi/2]$. Gli integrali ellittici completi $F(\pi/2, \alpha)$ ed $E(\pi/2, \alpha)$ si scrivono in maniera concisa rispettivamente come $F(\alpha)$ ed $E(\alpha)$.

È utile anche l'identità di Legendre:

$$E(\alpha)F(\pi/2-\alpha)+E(\pi/2-\alpha)F(\alpha)-F(\alpha)F(\pi/2-\alpha)=\pi/2,$$

e in particolare il caso speciale

$$2E(\pi/4)F(\pi/4)-(F(\pi/4))^2=\pi/2. \tag{2.26}$$

Approssimazione per angoli piccoli Dalla (2.24) è chiaro che

$$F(\psi,\alpha)=\psi+O(\alpha^2) \tag{2.27}$$

per $\alpha \to 0$.

Approssimazione per angoli grandi Sempre dalla (2.24), si ha:

$$F(\psi,\alpha)=F(\psi,\pi/2)+O(\pi/2-\alpha)^2, \tag{2.28}$$

uniformemente per $0 \le \psi \le \psi_0 < \pi/2$, per $\alpha \to \pi/2$. Inoltre, notiamo che

$$F(\psi,\pi/2)=\log\tan(\pi/4+\psi/2). \tag{2.29}$$

Trasformazione ascendente di Landen Se $0 < \alpha_i < \alpha_{i+1} < \pi/2$, $0 < \psi_{i+1} < \psi_i \le \pi/2$ sono tali che

$$\sin\alpha_i=\tan^2(\alpha_{i+1}/2), \tag{2.30}$$

e

$$\sin(2\psi_{i+1}-\psi_i)=\sin\alpha_i\sin\psi_i, \tag{2.31}$$

allora si ha:

$$F(\psi_{i+1},\alpha_{i+1})=[(1+\sin\alpha_i)/2]F(\psi_i,\alpha_i). \tag{2.32}$$

Se $s_i=\sin\alpha_i$ e $v_i=\tan(\psi_i/2)$, allora dalla (2.30) si ottiene

$$s_{i+1}=2s_i^{\frac{1}{2}}/(1+s_i), \tag{2.33}$$

e dalla (2.31) si ottiene

$$v_{i+1} = w_3/(1 + (1 + w_3^2)^{\frac{1}{2}}), \tag{2.34}$$

dove

$$w_3 = \tan\psi_{i+1} = (v_i + w_2)/(1 - v_i w_2), \tag{2.35}$$

$$w_2 = \tan(\psi_{i+1} - \psi_i/2) = w_1/(1 + (1 - w_1^2)^{\frac{1}{2}}), \tag{2.36}$$

$$w_1 = \sin(2\psi_{i+1} - \psi_i) = 2s_i v_i/(1 + v_i^2). \tag{2.37}$$

Definizione 7 *La relazione* (2.32), *che lega* $F(\alpha_{i+1}, \psi_{i+1})$ *e* $F(\alpha_i, \psi_i)$, *è detta* trasformazione ascendente di Landen.

2.3.2 Algoritmo AGM

Dalla trasformazione ascendente di Landen è possibile derivare la media aritmetico-geometrica di Gauss e Lagrange: se $a_0 = 1$, $b_0 = \cos\alpha > 0$, costruiamo le due successioni

$$a_{j+1} = (a_j + b_j)/2 \tag{2.38}$$

e

$$b_{j+1} = (a_j b_j)^{\frac{1}{2}}; \tag{2.39}$$

allora

$$\lim_{j\to+\infty} a_j = \pi/[2F(\alpha)]. \tag{2.40}$$

Inoltre, se $c_0 = \sin\alpha$ e

$$c_{j+1} = a_j - a_{j+1}, \tag{2.41}$$

allora

$$E(\alpha)/F(\alpha) = 1 - \sum_{j=0}^{+\infty} 2^{j-1} c_j^2. \tag{2.42}$$

Siano poi s_i, a_i e b_i come sopra, con $\alpha{=}\pi/2 - \alpha_0$, dimodoché $s_0 = a_0/b_0$. Dalle (2.33) e (2.38) segue che $s_i = b_i/a_i$ per ogni $i \geq 0$. Perciò, $(1+s_i)/2 = a_{i+1}/a_i$ e

$$\prod_{i=0}^{+\infty}[(1+s_i)/2] = \lim_{i\to+\infty} a_i = \pi/[2F(\pi/2-\alpha_0)] \tag{2.43}$$

segue dalla (2.40).

Un altro collegamento fra la (2.33) e la media aritmetico-geometrica è evidente se poniamo $s_0 = (1 - b_0^2/a_0^2)^{\frac{1}{2}}$. Supponendo che la (2.33) valga anche per $i < 0$, segue che $s_{-i} = (1 - b_i^2/a_i^2)^{\frac{1}{2}}$ per ogni $i \geq 0$. Questo può essere sfruttato per dedurre la (2.40) dalla (2.32).

2.3.3 Calcolo di π

Siano $a_0 = 1$, $b_0 = c_0 = 2^{-\frac{1}{2}}$, $A = \lim_{i\to+\infty} a_i$ e $T = \lim_{i\to+\infty} t_i$, dove a_i, b_i e c_i sono definiti dalle (2.38) e (2.41) per $i \geq 1$; inoltre, siano $t_i = \frac{1}{2}\sum_{j=0}^{i} 2^{j-1}c_j^2$. Dalle (2.26), (2.40) e (2.42), abbiamo che

$$\pi = A^2/T. \tag{2.44}$$

Poiché $a_i > b_0 > 0$ per ogni $i \geq 0$ e $c_{i+1} = a_i - a_{i+1} = a_{i+1} - b_i$, la seconda delle (2.38) ci dice che $b_{i+1} = [(a_{i+1}+c_{i+1})/(a_{i+1}-c_{i+1})]^{\frac{1}{2}} = a_{i+1} - O(c_{i+1}^2)$. Quindi, il procedimento converge con ordine almeno 2 e bastano $\log_2 n + O(1)$ iterazioni per stimare la (2.44) con un errore di $O(2^{-n})$. Un'analisi più dettagliata mostra che $a_{i+1}^2/t_i < \pi < a_i^2/t_i$ per ogni $i \geq 0$ e inoltre che $a_i^2/t_i - \pi \approx 8\pi\exp(-2^i\pi)$ e $\pi - a_{i+1}^2/t_i \approx \pi^2 2^{i+4}\exp(-2^{i+1}\pi)$ per $i \to +\infty$. Dall'analisi precedente è chiaro che il seguente algoritmo valuta π con precisione n:

```
A:=1; B:=2^{-1/2}; T:=1/4; X:=1;
while A-B>2^{-n} do
```

```
begin
    Y:=A; A:=(A+B)/2; B:=(BY)^{1/2};
    T:=T-X(A-Y)^2; X:=2X
end;
return A^2/T (o meglio, (A+B)^2/(4T))
```

Dal momento che servono $\log_2 n + O(1)$ iterazioni, si deve lavorare con precisione $n+O(\log\log n)$ anche se l'algoritmo è numericamente stabile. Ogni iterazione richiede $O(t_n(M))$ operazioni, quindi è possibile valutare π con precisione n in $O(t_n(M)\log n)$ operazioni. Ciò è asintoticamente più veloce dei metodi classici che impiegano $O(n^2)$ operazioni se si utilizza un'algoritmo di moltiplicazione rapido. Notare che, poiché l'iterazione aritmetico-geometrica non è un algoritmo autocorrettore, non è possibile ottenere un limite superiore di $O(t_n(M))$ così come è possibile per il calcolo del reciproco e della radice quadrata di un numero in virgola mobile tramite l'algoritmo di Newton.

2.3.4 Calcolo dell'esponenziale

Supponiamo che sia fissato un numero $\delta > 0$ e prendiamo $m \in [\delta, 1-\delta]$. Se $\sin\alpha_0 = m^{\frac{1}{2}}$, possiamo calcolare $F(\alpha_0)$ con precisione n in $O(t_n(M)\log n)$ operazioni, usando la (2.40) e l'iterazione aritmetico-geometrica. Applicando la trasformazione ascendente di Landen, con $i = 0, 1, \ldots, k-1$ e $\psi_0 = \pi/2$, si ha:

$$F(\psi_k, \alpha_k) = \prod_{i=0}^{k-1} [(1+\sin\alpha_i)/2] F(\alpha_0). \tag{2.45}$$

Poiché $s_0 = \sin\alpha_0 = m^{\frac{1}{2}} \geq \delta^{\frac{1}{2}} > 0$, dalla (2.33) segue che $s_i \to 1$ per $i \to +\infty$. Infatti, se $s_i = 1-\varepsilon_i$, allora $\varepsilon_{i+1} = 1-s_{i+1} = 1-2(1-\varepsilon_i)^{\frac{1}{2}}/(2-\varepsilon_i) = \varepsilon_i^2/8 + O(\varepsilon_i^3)$, perciò $s_i \to 1$ con ordine 2. Quindi, dopo $k \approx \log_2 n$ iterazioni abbiamo $\varepsilon_k = O(2^{-n})$, da cui $\pi/2 - \alpha_k = O(2^{-n/2})$ e, dalle (2.28) e (2.29), si

ha:

$$F(\psi_k, \alpha_k) = \log \tan(\pi/4 + \psi_k/2) + O(2^{-n}). \tag{2.46}$$

Supponendo $k > 0$, l'errore è uniformemente $O(2^{-n})$ per tutti gli $m \in [\delta, 1-\delta]$, poiché $\psi_k \leq \psi_1 < \pi/2$.

Definiamo le funzioni

$$U(m) = \left\{ \prod_{i=0}^{+\infty} [(1 + \sin \alpha_i)/2] \right\} F(\alpha_0) \tag{2.47}$$

e

$$T(m) = \tan(\pi/4 + \psi_\infty/2), \tag{2.48}$$

dove $\psi_\infty = \lim_{i \to +\infty} \psi_i$. Poiché $s_i \to 1$ con ordine 2, il prodotto infinito in (2.47) è convergente e $U(m)$ è analitico per tutti gli $m \in (0, 1)$. Prendendo il limite in (2.45) e (2.46) per n (e quindi k) che tende a $+\infty$, ricaviamo l'identità fondamentale

$$U(m) = \log T(m). \tag{2.49}$$

Usando le (2.33)–(2.37), calcoliamo $U(m) = \{\prod_{i=0}^{k-1} [(1 + s_i)/2]\} F(\alpha_0) + O(2^{-n})$ e $T(m) = (1+v_k)/(1-v_k) + O(2^{-n})$, a precisione n, in $O(t_n(M) \log n)$ operazioni, tramite i seguenti algoritmi:

Algoritmo per il calcolo di $U(m)$

```
A:=1; B:=(1-m)^(1/2);
while A-B>2^(-n/2) do
begin
    C:=(A+B)/2; B:=(AB)^(1/2); A:=C
end
A:=\pi/(A+B); S:=m^(1/2);
while 1-S>2^(-n/2) do
begin
```

```
    A:=A(1+S)/2; S:=2S^(1/2)/(1+S)
end;
return A(1+S)/2
```

Algoritmo per il calcolo di $T(m)$

```
V:=1; S:=m^(1/2);
while 1-S>2^(-n) do
begin
    W:=2SV/(1+V^2);
    W:=W/(1+(1-W^2)^(1/2));
    W:=(V+W)/(1-VW);
    V:=W/(1+(1+W^2)^(1/2));
    S:=2S^(1/2)/(1+S)
end;
return (1+V)/(1-V)
```

Dalla (2.43) e dalla (2.47),

$$U(m) = (\pi/2)F(\alpha_0)/F(\pi/2 - \alpha_0), \tag{2.50}$$

dove $\sin\alpha_0 = m^{\frac{1}{2}}$, come sopra. Sia $F(\alpha_0)$ che $F(\pi/2 - \alpha_0)$ possono essere calcolate mediante l'iterazione aritmetico-geometrica. Dalla (2.49) e dalla (2.50), si ricavano i casi speciali $U(\frac{1}{2}) = \pi/2$ e $T(\frac{1}{2}) = e^{\pi/2}$. Inoltre, la (2.50) dà:

$$U(m)U(1-m) = \pi^2/4, \tag{2.51}$$

per tutti gli $m \in (0, 1)$.

Sebbene sia meglio evitare valori di m vicini a 0 o 1, è interessante ricavare stime asintotiche di $U(m)$ e $T(m)$ per $m \to 0$ o 1. Dall'algoritmo per $T(m)$, $T(1-\varepsilon) = 4\varepsilon^{-\frac{1}{2}} - \varepsilon^{\frac{1}{2}} + O(\varepsilon^{\frac{3}{2}})$ per $\varepsilon \to 0$. Quindi, dalla (2.49), $U(1-\varepsilon) =$

$L(\varepsilon) - \varepsilon/4 + O(\varepsilon^2)$, dove $L(\varepsilon) = \log(4/\varepsilon^{\frac{1}{2}})$. Usando la (2.51), otteniamo $U(\varepsilon) = \pi^2/[4L(\varepsilon)] + O(\varepsilon/L^2)$, e quindi $T(\varepsilon) = \exp(\pi^2/[4L(\varepsilon)]) + O(\varepsilon/L^2)$.

Per il calcolo effettivo di $\exp x$ a precisione n, prima di tutto si usano le identità viste in precedenza per ridurre l'argomento all'interno di un dominio adatto, diciamo $1 \leq x \leq 2$. Indi, risolviamo l'equazione nonlineare

$$U(m) = x, \tag{2.52}$$

ottenendo m a precisione n, con un metodo adatto (ad esempio, il metodo di Newton). Questo può essere fatto in $O(t_n(M) \log n)$ operazioni. Dalla (2.49) e dalla (2.52), $T(m) = \exp x$, quindi abbiamo calcolato $\exp x$ a precisione n.

Capitolo 3

Calcolo della funzione logaritmo

Homo sum: nihil humani a me alienum puto.

TERENZIO

In questo capitolo presentiamo alcuni metodi per il calcolo del logaritmo naturale di un numero in virgola mobile. Questi metodi si basano in larga parte sugli algoritmi per il calcolo dell'esponenziale che abbiamo visto nel capitolo precedente.

3.1 La serie di Taylor

Anche qui, il metodo più elementare per calcolare il logaritmo naturale di un numero è tramite la serie di Taylor:

$$\log(1+x) = \sum_{j=0}^{+\infty} (-1)^{j-1} \frac{x^j}{j}. \tag{3.1}$$

L'errore R_k commesso arrestandosi al termine k-esimo non supera il primo termine trascurato: $R_k \leq \frac{|x|^{k+1}}{(k+1)}$.

Questa serie però converge solo per $|x| < 1$, in quanto altrimenti il suo termine generico non è infinitesimo. Una serie che permetta di calcolare il logaritmo naturale di ogni numero positivo è:

$$\log(\frac{1+x}{1-x}) = 2\sum_{j=0}^{+\infty} \frac{x^{2j+1}}{2j+1}, \tag{3.2}$$

che si ottiene sommando alla (3.1) la serie che si ottiene da essa sostituendovi $-x$ al posto di x; infatti, al variare di x nell'intervallo $(-1, 1)$, $\frac{1+x}{1-x}$ varia su tutto $\mathbb{R}^+$.

3.2 Il metodo dell'inversione dell'esponenziale

Un metodo elementare e in generale più rapido della serie di Taylor per calcolare il logaritmo x di un numero reale y è quello di risolvere con un metodo iterativo, ad esempio quello di Newton illustrato nel paragrafo 1.2.1, l'equazione $\exp(x) - y = 0$. Precisamente, scelto x_0 a piacere, il j-esimo passo iterativo è:

$$x_{j+1} = x_j - \frac{\exp(x_j) - y}{\exp(x_j)} = x_j - 1 + y\exp(-x_j),$$

e quindi il costo del calcolo è di un esponenziale e una moltiplicazione per passo. Il metodo ha ordine di convergenza 2, quindi è sufficiente effettuare le operazioni di ciascun passo con una precisione dimezzata rispetto al passo successivo. Se l'esponenziale viene calcolato in $O(t_n(M)\log n)$ operazioni, usando ad esempio il metodo AGM, allora anche il logaritmo viene così calcolato con un numero di operazioni dello stesso ordine di grandezza.

3.3 AGM

Il metodo AGM descritto nel paragrafo 2.3 riguardo al calcolo dell'esponenziale, può essere utilizzato anche per il calcolo del logaritmo. Riprendiamo qui le notazioni di quel paragrafo.

Il metodo AGM per il calcolo del logaritmo prevede di risolvere $T(m) = x$ (eventualmente dopo aver ridotto l'argomento con una delle trasformazioni usuali) e in seguito calcolare $U(m)$.

Se $x \in [1, 2]$, allora la soluzione m della (2.52) appartiene all'intervallo $(0.10, 0.75)$. Allo stesso modo, se $x \in [3, 9]$, la soluzione di $T(m) = x$ appartiene a $(0.16, 0.83)$.

Se $x = 1 + \varepsilon$, dove ε è piccolo, e si utilizza la relazione di riduzione

$$\log x = \log \lambda x - \log \lambda, \tag{3.3}$$

allora $\log \lambda x$ e $\log \lambda$ possono essere calcolati come sopra, ma gli errori di cancellazione nella (3.3) possono generare una perdita di precisione nel valore calcolato di $\log x$. Se $|\varepsilon| > 2^{-n}$, è sufficiente calcolare $\log \lambda x$ e $\log x$ con precisione $2n$, poiché si perdono al massimo n bit di precisione nella (3.3) per via della cancellazione. D'altro canto, non c'è difficoltà se $|\varepsilon| \leq 2^{-n}$, perché in questo caso $\log(1+\varepsilon) = \varepsilon(1+O(2^{-n}))$. Quando si calcola $\exp x$, non si verifica mai una simile perdita di precisione ed è sufficiente lavorare con precisione $n + O(\log \log n)$, come nel calcolo di π. Pertanto, anche $\log x$, così come $\exp x$, può essere calcolato con precisione n in $O(t_n(M) \log n)$ operazioni.

Capitolo 4

Calcolo di funzioni trigonometriche

Wir müssen wissen, wir werden wissen.

David Hilbert

Un'altra classe di funzioni di cui è interessante calcolare il valore in tempi rapidi è quella delle funzioni trigonometriche (sin, cos, tan) e le loro inverse. Anche per queste funzioni il calcolo può essere in molti casi ricondotto a quello del calcolo dell'esponenziale e quindi qui verranno riproposti, con le opportune modifiche, i metodi già presentati in precedenza. Viene inoltre presentata la classe di algoritmi CORDIC, utili per calcolare dette funzioni senza effettuare moltiplicazioni e quindi adatti ad essere implementati su circuiti molto semplici.

4.1 La serie di Taylor

Come già negli altri casi, il primo algoritmo preso in esame è quello che utilizza l'espansione in serie di Taylor di seno e coseno:

$$\sin x = \sum_{j=0}^{+\infty} (-1)^j \frac{x^{2j+1}}{(2j+1)!}, \tag{4.1}$$

$$\cos x = \sum_{j=0}^{+\infty} (-1)^j \frac{x^{2j}}{(2j)!}, \tag{4.2}$$

Queste espansioni sono legate alla serie di Taylor dell'esponenziale dalla formula di Eulero:

$$e^{ix} = \cos x + i \sin x. \tag{4.3}$$

Il calcolo anche qui può essere accelerato riducendo l'argomento ad un intervallo limitato, diciamo $[0, \pi/4]$ tramite varie identità trigonometriche. Un'ulteriore riduzione dell'argomento può essere ottenuta applicando ricorsivamente la formula di triplicazione del seno

$$\sin(3x) = (\sin x)(3 - (2 \sin x)^2). \tag{4.4}$$

Più precisamente, si può procedere in questo modo:

1. si divide k volte per 3 l'argomento iniziale in modo da ottenere $y = x/3^k$;

2. calcoliamo $\sin y$ tramite la somma di alcuni termini della sua serie di Taylor;

3. applicando ricorsivamente k volte la (4.4) al valore così trovato, otteniamo $\sin x$.

In particolare, si può utilizzare anche qui il metodo esposto nel paragrafo 2.1.2 per il calcolo dell'esponenziale; indi, si applica k volte la (4.4) per

ottenere $\sin x$. L'utilizzo della (4.4) richiede due moltiplicazioni a precisione piena per ciascuno dei k passi. L'argomento ridotto è dell'ordine di 3^{-k} e la serie di Taylor del seno ha la metà dei termini rispetto a quella dell'esponenziale. In questo caso, il numero totale di operazioni per il calcolo di $\sin x$ con n cifre di precisione è circa

$$W \approx \frac{n \log b}{2j(\log n + k \log 3)} + j + 2k,$$

dove b è la base dell'aritmetica utilizzata.

Come in precedenza, cerchiamo j e k che minimizzino W; in questo caso, si trova:

$$\begin{aligned} j &= n^{1/3}(\log b/\log 3)^{1/3}, \\ k &= \frac{1}{2}n^{1/3}(\log b/\log 3)^{1/3} - \log n/\log 3, \end{aligned}$$

e pertanto è possibile calcolare $\sin x$ in un tempo $O(n^{1/3}t_n(M))$. Poiché la serie di Taylor ha solo $t/2$ termini e l'inversione della riduzione dell'argomento comporta il doppio delle operazioni rispetto all'esponenziale, l'algoritmo esegue un numero minore di passi di riduzione rispetto al calcolo di $\exp x$.

Le altre funzioni trigonometriche possono essere calcolate a partire da $\sin x$ e da varie identità. In particolare, per il calcolo della tangente trigonometrica, si può far ricorso alle identità:

$$\tan x = \frac{\sin x}{\cos x} = \frac{\sin x}{\pm\sqrt{1 - \sin^2 x}} = \frac{\pm\sqrt{1 - \cos^2 x}}{\cos x}$$

che consentono di calcolarla tramite il calcolo di seno e coseno (o di una delle due funzioni e una radice) e una divisione.

Le funzioni trigonometriche inverse si possono calcolare partendo da quelle dirette e utilizzando il metodo di Newton. Facciamo vedere il procedimento per calcolare $x = \arcsin y$ risolvendo l'equazione $\sin x = y$. Anche qui vediamo il passo induttivo del metodo, che si applica dopo aver scelto il punto

iniziale x_0:

$$x_{j+1} = x_j - \frac{\sin(x_j) - y}{\cos(x_j)}$$

che comporta, ad ogni passo, il calcolo di un seno, un coseno e una divisione. Anche qui, essendo l'ordine di convergenza del metodo almeno 2, ogni operazione può essere svolta in precisione dimezzata rispetto al paso seguente.

In questo modo, tutte le funzioni trigonometriche si possono calcolare in un tempo di $O(n^{1/3}t_n(M))$.

4.2 Le approssimanti di Padé

Il metodo descritto nel paragrafo 2.2 è adattabile anche per il calcolo delle funzioni trigonometriche, in particolare di seno e coseno. Presentiamo qui il metodo per il calcolo del seno: per le altre funzioni, il procedimento è analogo.

Nello specifico, si può effettuare una prima riduzione dell'argomento in questo modo: se s è l'argomento del seno, si pone:

$$\frac{4}{\pi}|s| = m + t, \quad m \in \mathbb{N}, \quad t \in [0, 1);$$

allora si dimostra che

$$\sin s = \operatorname{sgn} s \cdot (-1)^{\lfloor m/4 \rfloor} \cdot \begin{cases} \sin \frac{\pi}{4}t, & m \equiv 0, \\ \cos \frac{\pi}{4}(1-t), & m \equiv 1, \\ \cos \frac{\pi}{4}t, & m \equiv 2, \\ \sin \frac{\pi}{4}(1-t), & m \equiv 3 \end{cases} \pmod 4,$$

dove $\operatorname{sgn} s$ è la funzione segno. L'argomento viene quindi ridotto all'intervallo $[0, 1)$. Un'ulteriore riduzione è possibile tenendo presente la (4.4)

A questo punto, utilizzando la (4.3), si può esprimere $\sin x$ in funzione di $\alpha(x)$ e $\beta(x)$ definite nel paragrafo 2.2:

$$\sin x = \Im(e^{ix}) = \frac{2x\alpha(ix)\beta(ix)}{\alpha^2(ix) + x^2\beta^2(ix)},$$

dove $\Im(z)$ è la parte immaginaria del numero complesso z; quindi, tramite il calcolo di $\alpha(x)$ e $\beta(x)$ è possibile calcolare $\sin x$ analogamente a quanto già fatto per $\exp x$.

Le considerazioni sul costo computazionale descritte nel paragrafo 2.2 valgono invariate, a condizione di sostituire $b_1 = 1$ e $b_2 = 0$ con $b_1 = b_2 = 2$ ed $\eta = 2$ con $\eta = 3$ per tenere conto del fatto per la riduzione dell'argomento si utilizza la (4.4) invece della (2.2).

4.3 AGM

Anche questo metodo, descritto per il calcolo dell'esponenziale, è implementabile per il calcolo delle funzioni trigonometriche.

Fissiamo $\delta > 0$ e sia $x \in [\delta, 1]$. Sia $s_0 = \sin \alpha_0 = 2^{-n/2}$ e $v_0 = \tan(\psi_0/2) = x/(1 + (1 + x^2)^{\frac{1}{2}})$, dimodoché $\tan \psi_0 = x$. Applicando la trasformazione ascendente di Landen (2.33), si ha:

$$F(\psi_k, \alpha_k) = \{\prod_{j=0}^{k-1}[(1 + s_j)/2]\}F(\psi_0, \alpha_0). \tag{4.5}$$

Inoltre, dalla (2.27) e per la scelta fatta di s_0, si ha:

$$F(\psi_0, \alpha_0) = \arctan x + O(2^{-n}). \tag{4.6}$$

Dalla (2.33), $s_{i+1} \geq s_i^{\frac{1}{2}}$, quindi esiste $j \leq \log_2 n + O(1)$ tale che $s_j \in [\frac{1}{4}, \frac{4}{5}]$. Poiché $s_i \to 1$ con ordine 2, esiste $k \leq 2\log_2 n + O(1)$ tale che $1 - s_k = O(2^{-n})$.

Dalle (2.28)–(2.29), $F(\psi_k, \alpha_k) = \log\tan(\pi/4 + \psi_k/2) + O(2^{-n})$. Perciò, dalle (4.5)–(4.6),

$$\arctan x = \{\prod_{j=0}^{k=1}[2/(1+s_i)]\}\log\tan(\pi/4+\psi_k/2) + O(2^{-n}). \qquad (4.7)$$

Se calcoliamo $\tan(\pi/4 + \psi_k/2)$ come sopra, ed usiamo le considerazioni del paragrafo precedente per calcolare il logaritmo in (4.7), possiamo ottenere $\arctan x$ a precisione n in $O(t_n(M)\log n)$ operazioni. L'algoritmo può essere scritto in questo modo:

Algoritmo per il calcolo di $\arctan x$

```
S:=2^(-n/2); V:=x/(1+(1+x^2)^(1/2)); Q:=1;
while 1-s>2^(-n) do
begin
    Q:=2Q/(1+S);
    W:=2SV/(1+V^2);
    W:=W/(1+(1-W^2)^(1/2));
    W:=(V+W)/(1-VW);
    V:=W/(1+(1+W^2)^(1/2));
    S:=2S^(1/2)/(1+S)
end;
return Q log((1+V)/(1-V))
```

Dopo k iterazioni, $Q < 2^{-k}$ e quindi si perdono al massimo $2\log_2 n + O(1)$ bit di precisione perché V è piccolo. Perciò è sufficiente lavorare con precisione $n + O(\log n)$, quindi bastano $O(t_n(M)\log n)$ operazioni per ottenere $\arctan x$ con precisione n.

4.4 L'algoritmo CORDIC

CORDIC (acronimo di *COordinate ROtation DIgital Computer*) è una classe di algoritmi che consente di calcolare le funzioni trigonometriche tramite rotazioni calcolabili (in aritmetica binaria) solo con *shift* e addizioni. Questi algoritmi sono stati creati per essere implementati su circuiti che non hanno un moltiplicatore hardware implementato; in caso contrario, quando cioè si hanno a disposizione istruzioni moltiplicative integrate nel circuito, gli algoritmi descritti in precedenza sono più vantaggiosi.

L'algoritmo qui descritto [16, 17] si basa sul sistema di equazioni:

$$x' = x\cos\phi - y\sin\phi$$
$$y' = y\cos\phi + x\sin\phi$$

che ruota il vettore (x, y) nel piano cartesiano di un angolo ϕ. Questo sistema può essere riscritto:

$$x' = \cos\phi[x - y\tan\phi]$$
$$y' = \cos\phi[y + x\tan\phi].$$

Se ora si sceglie un angolo ϕ in modo che sia $\tan\phi = \pm 2^{-k}$, la moltiplicazione per $\tan\phi$ si riduce ad un semplice *shift*. In questo modo è possibile ruotare il vettore di un angolo arbitrario ϕ per mezzo di rotazioni successive per angoli più piccoli ϕ_k. Se ad ogni passo si sceglie soltanto il verso della rotazione, allora il termine $\cos\phi$ fuori parentesi diventa una costante indipendente da esso (in quanto, per ogni angolo α, $\cos(-\alpha) = \cos\alpha$). L'iterazione che permette di calcolare la rotazione può allora essere espressa così:

$$x_{k+1} = C_k[x_k - y_k \cdot d_k \cdot 2^{-k}]$$
$$y_{k+1} = C_k[y_k + x_k \cdot d_k \cdot 2^{-k}]$$

dove

$$C_k = \cos \arctan 2^{-k} = \frac{1}{\sqrt{1 + 2^{-2k}}}$$
$$d_k = \pm 1.$$

Osserviamo che possiamo rimuovere i C_k dalle formule di rotazione e raccoglierli in un unico fattore in modo da ottenere un algoritmo che utilizzi solo *shift* ed addizioni. Inoltre, poiché

$$\prod_{k=0}^{+\infty} \frac{1}{\sqrt{1 + 2^{-2k}}} \approx 0.6073,$$

alla fine dei passi di rotazione possiamo moltiplicare il risultato per 0.6073 per ottenere il risultato desiderato (il valore esatto del fattore moltiplicativo dipende dal numero effettivo di rotazioni effettuate).

L'angolo di rotazione è definito in maniera univoca dalla successione delle direzioni delle rotazioni elementari. Questa successione può essere rappresentata in un vettore, detto *di decisione*. L'insieme di tutti i possibili vettori rappresenta un sistema di misura degli angoli. È possibile effettuare delle conversioni da questo sistema in altri e viceversa usando delle tabelle di conversione. Un metodo migliore per convertire gli angoli è quello di usare un terzo valore che accumuli i valori delle rotazioni elementari ad ogni passo. Questo accumulatore aggiunge una terza equazione al sistema di rotazione:

$$z_{k+1} = z_k - d_k \cdot \arctan 2^{-k}.$$

Gli algoritmi CORDIC sono normalmente divisi in due categorie: algoritmi di rotazione e algoritmi di vettorizzazione. Nella modalità di rotazione, l'accumulatore viene inizializzato con l'angolo di cui si vuol far ruotare il vettore; ad ogni passo, la direzione di rotazione viene scelta in modo da

minimizzare il modulo dell'angolo residuo nell'accumulatore. Ad ogni iterazione, la decisione viene quindi presa guardando il segno di z_k. Le equazioni dell'algoritmo CORDIC nel caso rotazionale sono quindi:

$$\begin{aligned} x_{k+1} &= x_k - y_k \cdot d_k \cdot 2^{-k} \\ y_{k+1} &= y_k + x_k \cdot d_k \cdot 2^{-k} \\ z_{k+1} &= z_k - d_k \cdot \arctan 2^{-k}, \end{aligned}$$

dove

$$d_k = -1 \text{ se } z_k < 0, \ +1 \text{ altrimenti},$$

che danno alla fine:

$$\begin{aligned} x_n &= A_n[x_0 \cos z_0 - y_0 \sin z_0] \\ y_n &= A_n[y_0 \cos z_0 + x_0 \sin z_0] \\ z_n &= 0 \\ A_n &= \prod_{k=0}^{n} \sqrt{1 + 2^{-2k}}. \end{aligned}$$

Nella modalità vettore, l'algoritmo ruota il vettore dell'angolo necessario ad allinearlo con l'asse x. Il risultato dell'operazione è l'angolo di rotazione e il modulo del vettore originale (memorizzato nella componente x del risultato) eventualmente scalata di un fattore pari al prodotto dei C_k. Questo algoritmo cerca di minimizzare la componente y del vettore residuo ad ogni rotazione. Il segno della componente y residua viene utilizzato per decidere la direzione di rotazione al passo successivo. Se l'accumulatore z_0 viene inizializzato a zero, al termine dell'algoritmo z_n conterrà l'angolo di cui è stato ruotato il vettore iniziale. Pertanto, le equazioni dell'algoritmo CORDIC nel

caso vettoriale sono:

$$x_{k+1} = x_k - y_k \cdot d_k \cdot 2^{-k}$$
$$y_{k+1} = y_k + x_k \cdot d_k \cdot 2^{-k}$$
$$z_{k+1} = z_k - d_k \cdot \arctan 2^{-k},$$

dove

$$d_k = -1 \text{ se } y_k < 0, \ +1 \text{ altrimenti},$$

che danno alla fine:

$$x_n = A_n \sqrt{x_0^2 + y_0^2}$$
$$y_n = 0$$
$$z_n = z_0 + \arctan \frac{y_0}{x_0}$$
$$A_n = \prod_{k=0}^{n} \sqrt{1 + 2^{-2k}}.$$

A questo punto, mediante gli algoritmi suesposti, siamo in grado di calcolare le funzioni trigonometriche e trigonometriche inverse.

4.4.1 Calcolo di sin e cos

L'algoritmo CORDIC in modalità rotazionale può calcolare simultaneamente il seno e il coseno dell'angolo iniziale. Ponendo uguale a zero la componente y del vettore di partenza, otteniamo:

$$x_n = A_n \cdot x_0 \cos z_0$$
$$y_n = A_n \cdot x_0 \sin z_0.$$

Ponendo $x_0 = 1/A_n$, la rotazione fornisce $\sin z_0$ e $\cos z_0$ senza fattori di scala.

4.4.2 Calcolo di arctan

L'arcotangente è calcolata in maniera diretta dall'algoritmo CORDIC in modalità vettoriale se l'accumulatore dell'angolo viene inizializzato a zero. L'argomento iniziale deve essere fornito come rapporto, espresso come vettore (x, y). Ciò ha il vantaggio di poter rappresentare anche un valore infinito, ponendo $x = 0$. Dal momento che l'arcotangente è memorizzata nell'accumulatore, il fattore di scala dell'algoritmo non inficia il risultato.

4.4.3 Calcolo di arcsin e arccos

L'arcoseno può essere calcolato partendo da un vettore unitario lungo il semiasse delle x positive e ruotandolo finché la componente y non sia uguale all'argomento c della funzione. A questo punto, $\arcsin c$ è l'angolo di cui si è ruotato il vettore, che è contenuto, come in precedenza, nell'accumulatore z. Il vettore di decisione, in questo caso, viene formato per confronto di c con la componente y_k del vettore ruotato ad ogni passo. L'algoritmo è pertanto:

$$\begin{aligned} x_{k+1} &= x_k - y_k \cdot d_k \cdot 2^{-k} \\ y_{k+1} &= y_k + x_k \cdot d_k \cdot 2^{-k} \\ z_{k+1} &= z_k - d_k \cdot \arctan 2^{-k}, \end{aligned}$$

dove

$$\begin{aligned} d_k &= +1 \text{ se } y_k < c, \ -1 \text{ altrimenti} \\ c &= \text{argomento iniziale} \end{aligned}$$

che alla fine produce:

$$\begin{aligned} x_n &= \sqrt{(A_n \cdot x_0)^2 - c^2} \\ y_n &= c \\ z_n &= z_0 + \arcsin \frac{c}{A_n \cdot x_0} \\ A_n &= \prod_{k=0}^{n} \sqrt{1 + 2^{-2k}}. \end{aligned}$$

e quindi, come in precedenza, se $x_0 = 1/A_n$ e $z_0 = 0$, si ha $z_n = \arcsin c$.

L'algoritmo fornisce valori corretti per ogni $c \in [-1, 1]$; comunque, a causa del fattore di scala, per valori di c molto prossimi a ± 1 l'algoritmo diventa molto poco accurato e il vettore ottenuto alla fine della ricorsione ha lunghezza inferiore a quello iniziale, pertanto alcuni elementi del vettore di decisione potrebbero essere scorretti.

La funzione arcocoseno si calcola in maniera simile alla precedente, partendo da un vettore parallelo all'asse y e ruotando finché $x_n = c$. Un altro modo per calcolare questa funzione è calcolare $\arcsin c$ e sottrarre $\pi/2$ dal risultato, sfruttando la nota indentità $\arcsin c + \arccos c = \pi/2$. Sarà necessario un cambio di segno se l'angolo risultante si trovasse nel quarto quadrante.

4.4.4 Complessità computazionale

La convergenza del metodo è lineare, in quanto vengono aggiunti in media 2 bit di precisione per passo. Pertanto, questo metodo, pur non necessitando di moltiplicazioni ma solo di addizioni e *shift*, è un metodo a convergenza lenta. Questo è il motivo per cui il metodo CORDIC è vantaggioso solo quando non si disponga di un moltiplicatore hardware integrato, mentre negli altri casi si preferisce usare metodi classici a convergenza più rapida.

Bibliografia

[1] Richard P. Brent. A FORTRAN multiple-precision arithmetic package. *ACM Trans. Math. Software*, 4:57–70, 1978.

[2] David H. Bailey. A portable high performance multiprecision package. Rapporto Tecnico RNR-90-022, NASA Ames Research Center, Moffett Field, CA 94035, 1992.

[3] The GNU MP Bignum library. `http://www.swox.com/gmp/`.

[4] Arnold Schönhage e Volker Strassen. Schnelle Multiplikation grosser Zahlen. *Computing*, 7:281–292, 1971.

[5] Richard P. Brent. Fast multiple-precision evaluation of elementary functions. *JACM*, 23(2):242–251, 1976.

[6] David M. Smith. Efficient multiple-precision evaluation of elementary functions. *Mathematics of Computation*, 52(185):131–134, 1989.

[7] Xu Guo-liang e Li Jia-kai. Evaluation of elementary functions for variable precision. *J. Num. Meth. & Comp. Appl.*, 15(3):161–171, 1994.

[8] Donald E. Knuth. *The Art of Computer Programming*, volume 2. Addison-Wesley, 1997.

[9] Xavier Gourdon e Pascal Sebah. FFT based multiplication of large numbers. `http://numbers.computation.free.fr/Constants/Algorithms/fft.ps`.

[10] Roberto Bevilacqua, Dario Bini, Milvio Capovani, e Ornella Menchi. *Metodi Numerici*. Zanichelli, 1997.

[11] William Feller. *An Introduction to Probability Theory and its Applications*. Wiley, 1960.

[12] V.Y. Pan. On methods of computing values of polynomials. *Russian Math. Surveys*, 21:105–136, 1966.

[13] Paolo Botti. *Il calcolo della funzione esponenziale in multiprecisione con il pacchetto GMP*. Tesi di laurea, Università di Pisa, 1996/97.

[14] George A. Baker e Peter Graves-Morris. *Padé Approximants*. Cambridge University Press, 1996.

[15] Milton Abramowitz e Irene A. Stegun. *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*. Dover, 1965.

[16] Ray Andraka. A survey of CORDIC algorithms for FPGA based computers. `http://www.andraka.com/files/crdcsrvy.pdf`, 1998.

[17] Ken Turkowski. *Graphics Gems I*, capitolo Fixed-Point Trigonometry with CORDIC Iterations, pagine 494–497. Academic Press, 1990. URL `http://www.worldserver.com/turk/computergraphics/FixedPointTrigonometry.pdf`.