Calcolare una potenza con esponente reale negativo

github.com/nrizzo

7 maggio 2018

1 Studio del problema

Si tratta del problema della potenza dal punto di vista computazionale con precisione arbitraria, in particolare con base 2 ed esponente reale negativo; in simboli, si vuole calcolare $f(x) = 2^{-x}$, dato $x \geq 0$.

2 Nozioni di calcolo scientifico

Riprendendo nozioni di calcolo scientifico, l'errore relativo commesso nel calcolo di una funzione $f(x)$ può essere considerato come la combinazione di tre diverse componenti:

$$\epsilon_{\text{inerente}} + \epsilon_{\text{analitico}} + \epsilon_{\text{algoritmico}} = \frac{f(x) - f(\tilde{x})}{f(x)} + \frac{f(\tilde{x}) - g(\tilde{x})}{f(\tilde{x})} + \frac{g(\tilde{x}) - \tilde{g}(\tilde{x})}{g(\tilde{x})}$$

Errore inerente L'errore inerente è dovuto ad un'approssimazione del dato in ingresso x ed è tale che

$$|\epsilon_{\text{inerente}}| \leq \left|\text{cond}_f(x)\right| \cdot \epsilon_x \quad \text{e} \quad \text{cond}_f(x) = \frac{|x| \cdot |f'(x)|}{|f(x)|};$$

ponendo $f(x) = 2^{-x}$,

$$\text{cond}_f(x) = \frac{|x| \cdot \left| -\log 2 \cdot 2^{-x} \right|}{\left|2^{-x}\right|} = |x| \log 2,$$

cioè il problema è mal condizionato per valori di $|x|$ grandi.

Errore analitico L'errore analitico deriva dalla funzione approssimata effettivamente calcolata.

Errore algoritmico L'errore algoritmico deriva dall'introduzione e la propagazione di errori nell'effettivo calcolo della funzione, ad esempio con le operazioni di macchina.

3 Metodo ingenuo

Rilassando la richiesta di avere precisione arbitraria, come può essere trattata l'operazione dall'aritmetica di macchina? La normale implementazione di `pow` delle librerie standard dovrebbe essere costante e usare una manciata di istruzioni `x86`, tra cui probabilmente c'è l'istruzione `F2XM1`; l'errore introdotto da questa è definito nella documentazione.[1][2]

4 Serie di MacLaurin

Per il calcolo di $f(x) = 2^{-x}$, con $x \geq 0$, si può separare x nella sua parte intera e frazionaria, tali che $x = \lfloor x \rfloor + \{x\}$, con $0 \leq \{x\} \leq 1$. Il calcolo della funzione diventa

$$f(x) = 2^{-\lfloor x \rfloor} \cdot 2^{-\{x\}};$$

il primo fattore è facilmente ottenibile, mentre il secondo viene calcolato con la serie di MacLaurin di 2^{-x}.

4.1 La serie

La serie di MacLaurin per $f(x) = 2^{-x}$ è

$$\begin{aligned} T(x) &= 1 - \log 2 \cdot x + \frac{\log^2 2}{2} \cdot x^2 - \frac{\log^3 2}{3!} \cdot x^3 + \frac{\log^4 2}{4!} \cdot x^4 + \dots \\ &= \sum_{i=0}^{+\infty} (-1)^i \cdot \frac{(\log 2 \cdot x)^i}{(i)!}. \end{aligned}$$

Approssimando $f(x)$ con il polinomio di MacLaurin di grado n, cioè

$$T_n(x) = \sum_{i=0}^{n} (-1)^i \cdot \frac{(\log 2 \cdot x)^i}{(i)!},$$

si ha che $f(x) = T_n(x) + R_n(x)$, con R_n il resto nella forma di Lagrange [3, p. 229]; esso equivale a

$$R_n(x) = (-1)^{n+1} \cdot \frac{\log^{n+1} 2 \cdot 2^{-\xi}}{(n+1)!} \cdot x^{n+1}, \quad \xi \in (0, x).$$

È facile vedere che $f(y) = 2^{-y}$ è una funzione decrescente, potendo così ottenere una stima per eccesso dell'errore (assoluto) commesso nell'approssimazione della funzione:

$$\big|f(x) - T_n(x)\big| = \big|R_n(x)\big| \leq \frac{(\log 2 \cdot x)^{n+1}}{(n+1)!}.$$

4.2 Implementazione

Sfruttando i risultati della sezione precedente è stato implementato in C del codice che, dato x e la tolleranza dell'errore analitico, divide x in parte intera e parte frazionaria e individua (per tentativi) il minimo grado n del polinomio di

MacLaurin per calcolare $2^{-\{x\}}$ in modo da garantire un errore più piccolo della tolleranza; poi calcola $T_n(x)$.[1]

```c
double reciprocal_exp2(double x, double tol)
{
        double fl = floor(x); /* parte intera */
        double fr = x - fl; /* parte frazionaria */

        /* 2^(-x) = 2^(-fl)*2^(-fr) */
        return pow(2,-fl) * reciprocal_exp2_aux(fr,tol);
}
```

```c
double reciprocal_exp2_aux(double x, double maxtol)
{
        double res;
        double log2x = log(2)*x;

        res = 0;
        for (int i = calcn(x, maxtol); i > 0; i--) {
                if (i%2 == 0)
                        res += 1;
                else
                        res += -1;

                res *= log2x;
                res /= i;
        }
        res += 1;

        return res;
}
```

```c
int calcn(double x, double tol)
{
        int n = 0;
        double maxerr = log(2)*x;

        while (tol < maxerr) {
                n++;
                maxerr *= (log(2)*x)/(n+1);
        }

        return n;
}
```

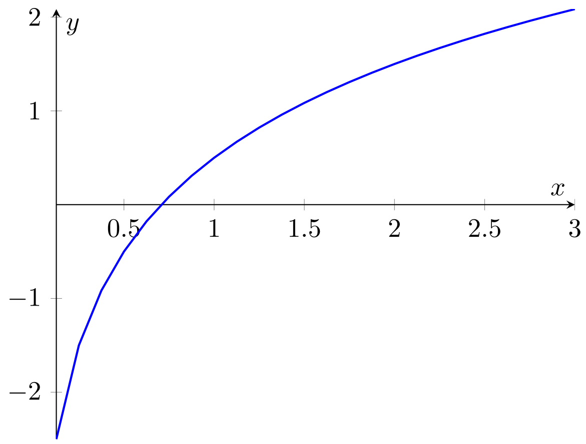
[1] L'implementazione usa numeri di macchina con precisione doppia ed è perciò puramente dimostrativa, visto che `pow` (della libreria `math`) calcola già la funzione con precisione massima! Se si volesse implementare l'algoritmo su un tipo di dato diverso, bisognerebbe avere già somma, moltiplicazione, divisione, pavimento, potenza di 2 con esponente negativo ma intero e logaritmo di 2.

5 Zeri di funzione

Se avessimo già un buon modo per calcolare il logaritmo, si potrebbe ricercare lo zero della seguente funzione, ad esempio con il metodo delle tangenti:

$$h(y) = \log_2 y + x, \quad x \geq 0.$$

Grafico di $\log_2 x + 0{,}5$



Riferimenti bibliografici

[1] *Implementazioni in aritmetica di macchina di `pow(x,y)` con `x`, `y` reali*, https://stackoverflow.com/questions/4638473/how-to-powreal-real-in-x86

[2] *Intel® 64 and IA-32 architectures software developer's manual combined volumes: 1, 2A, 2B, 2C, 2D, 3A, 3B, 3C, 3D, and 4*, Sezione 8.3.10 del Volume 1, https://software.intel.com/sites/default/files/managed/39/c5/325462-sdm-vol-1-2abcd-3abcd.pdf

[3] *Analisi matematica*, Seconda Edizione, Bertsch, Dal Passo, Giacomelli, McGraw-Hill, 2011.