# Sul calcolo approssimato di $\sqrt[k]{a}$

Stefano Costa

14 maggio 2012

Queste pagine descrivono un metodo per calcolare $\sqrt[k]{a}$ in modo approssimato utilizzando le quattro operazioni elementari; il valore ricavato al primo tentativo è in genere abbastanza accurato da consentire di raggiungere una precisione notevole attraverso poche iterazioni di approssimazione successive. Nella pratica il calcolo manuale è applicabile per radici quadrate o cubiche, ma il metodo verrà esposto nella sua generalità, con esempi per radici di ordine superiore.

Consideriamo lo sviluppo di Taylor per $(1+x)^\alpha$, $|x| < 1$:

$$(1+x)^\alpha = \sum_{m=0}^{\infty} \binom{\alpha}{m} x^m \overset{x\to 0}{\approx} 1 + \alpha x \tag{1}$$

per cui si ha ad esempio $\sqrt[3]{1.1} \approx 1 + \frac{0.1}{3} = 1.0333$, il valore corretto essendo 1.0323. Va detto per inciso che dall'approssimazione considerata è possibile ricavare l'iterazione di Newton: ponendo $\alpha = 1/k$ si ha

$$\begin{aligned} x_{n+1} \quad \approx \quad \sqrt[k]{a} &= \sqrt[k]{a + x_n^k - x_n^k} = x_n \sqrt[k]{1 + \frac{a - x_n^k}{x_n^k}} \approx x_n + \frac{a - x_n^k}{k x_n^{k-1}} \\ &= \frac{k-1}{k} x_n + \frac{1}{k} \frac{a}{x_n^{k-1}} \end{aligned}$$

Mettiamo subito in evidenza che $|x| < 1 \Rightarrow 0 < 1 + x < 2$, pertanto operativamente è necessario portarsi in queste condizioni: volendo calcolare $\sqrt{3} = 1.732$ *non* si scriverà $\sqrt{3} = \sqrt{1+2} \approx 1 + 2/2 = 2$, bensì

$$\sqrt{3} = \sqrt{4 \cdot 3/4} = 2\sqrt{0.75} = 2\sqrt{1 - 0.25} \approx 2(1 - 0.25/2) = 1.75$$

Detti ora $\xi = \sqrt[k]{a}$ ed $\epsilon$ una (piccola) perturbazione della soluzione $\xi$, utilizzando ancora una volta la (1) si ha:

$$x_n^k = (\xi(1+\epsilon_n))^k = \xi^k (1+\epsilon_n)^k = a(1+\epsilon_n)^k \overset{\epsilon_n \to 0}{\approx} a(1 + k\epsilon_n)$$

e pertanto

$$\epsilon_n \approx (x_n^k/a - 1)/k$$

Come vedremo dagli esempi, solitamente il valore di primo tentativo $x_0$ è sufficientemente accurato, ossia $\epsilon_0$ è sufficientemente piccolo, da consentire di raggiungere una precisione di 6 cifre significative in sole 2 o 3 iterazioni di Newton.

Il metodo che ci apprestiamo ora a descrivere è ispirato da un report[1] di K. Turkowski per il calcolo della radice cubica. Brevemente, si parte dall'osservazione che

$$\sqrt[k]{a} = \sqrt[k]{2^{kp}b} = 2^p \sqrt[k]{b}$$

e pertanto la radice $k-$esima di qualsiasi numero è in relazione semplice con quella di un numero $2^{kp}$ più piccolo: in altre parole, calcolare la radice $k-$esima di qualsiasi numero può ricondursi a calcolare quella di un numero $\frac{1}{2^{k-1}} \leq b < 2$. I passi per il calcolo della radice $k-$esima sono dunque i seguenti:

- si porta fuori radice un'eventuale potenza $k-$esima di 10; questo passaggio non è richiesto in una implementazione su calcolatore
- se $1 + x \geq 2$, si porta fuori radice la minima potenza $k-$esima di 2 che consente di applicare la (1). La scelta della base 2 è giustificata dal fatto che è relativamente semplice calcolarne le potenze anche a mente; inoltre, come si vedrà in seguito, per come i numeri vengono rappresentati internamente al calcolatore, è immediato eliminarne le potenze con un'operazione di resto modulo $k$
- si utilizza la (1) per approssimare la radice
- volendo, si migliora l'approssimazione con l'algoritmo di Newton
- si esegue il prodotto dei termini trovati

Vediamo alcuni esempi pratici. Approssimiamo a 6 cifre significative per mostrare l'accuratezza del valore di primo tentativo di $x_0$; nella pratica non si va oltre le 2 o 3 cifre.

1. si desideri approssimare $\sqrt[3]{100} = 4.64159$. Si ha:

$$\begin{aligned} x_0 &= \sqrt[3]{100} = \sqrt[3]{2^6 \frac{100}{64}} = 2^2 \sqrt[3]{1.5625} \\ &\approx 4\left(1 + \frac{0.5625}{3}\right) = 4 \cdot 1.1875 = 4.75 \end{aligned}$$

   ove $\epsilon_0 = (1.1875^3/1.5625 - 1)/3 = 2.39\%$. Volendo migliorare la soluzione con il metodo di Newton, si ha $1/k = 0.333$ e $(k-1)/k = 0.667$ e

$$x_1 = 4\left(0.667 \cdot 1.1875 + 0.333 \frac{1.5625}{1.1875^2}\right) = 4 \cdot 1.16104 = 4.64415$$

$$x_2 = 4\left(0.667 \cdot 1.16175 + 0.333 \frac{1.5625}{1.16175^2}\right) = 4 \cdot 1.1604 = 4.64159$$

2. si desideri approssimare $\sqrt[3]{343000} = 70$. Si ha:

$$\begin{aligned} x_0 &= \sqrt[3]{343000} = 10 \sqrt[3]{2^9 \frac{343}{512}} = 10 \cdot 2^3 \sqrt[3]{0.669922} \\ &\approx 80\left(1 - \frac{0.330078}{3}\right) = 80 \cdot 0.889974 = 71.1979 \end{aligned}$$

[1] K. Turkowski, *Computing the Cube Root*, Apple Computer Technical Report #KT-32, 1998

ove $\epsilon_0 = (0.889974^3/0.669922 - 1)/3 = 1.74\%$. Volendo migliorare la soluzione con il metodo di Newton, si ha ancora $1/k = 0.333$ e $(k-1)/k = 0.667$ e

$$x_1 = 80\left(0.667 \cdot 0.889974 + 0.333\frac{0.669922}{0.889974^2}\right) = 80 \cdot 0.875265 = 70.0212$$

$$x_2 = 80\left(0.667 \cdot 0.875265 + 0.333\frac{0.669922}{0.875265^2}\right) = 80 \cdot 0.875000 = 70$$

3. si desideri approssimare $\sqrt{\pi} = 1.77245$. Si ha:

$$\begin{aligned} x_0 &= \sqrt{\pi} = \sqrt{2^2\frac{\pi}{4}} = 2\sqrt{0.785398} \\ &\approx 2\left(1 - \frac{0.214602}{2}\right) = 2 \cdot 0.892699 = 1.7854 \end{aligned}$$

ove $\epsilon_0 = (0.892699^2/0.785398 - 1)/2 = 0.73\%$. Volendo migliorare la soluzione con il metodo di Newton, si ha $1/k = (k-1)/k = 0.5$ e

$$x_1 = 2\left(0.5 \cdot 0.892688 + 0.5\frac{0.785398}{0.892688}\right) = 2 \cdot 0.886250 = 1.77250$$

$$x_2 = 2\left(0.5 \cdot 0.88625 + 0.5\frac{0.785398}{0.88625}\right) = 2 \cdot 0.886227 = 1.77245$$

4. si desideri approssimare $\sqrt[5]{55} = 2.22881$. Si ha:

$$\begin{aligned} x_0 &= \sqrt[5]{55} = \sqrt[5]{2^5\frac{55}{32}} = 2\sqrt[5]{1.71875} \\ &\approx 2\left(1 + \frac{0.71875}{5}\right) = 2 \cdot 1.14375 = 2.2875 \end{aligned}$$

ove $\epsilon_0 = (1.14375^5/1.71875 - 1)/5 = 2.77\%$. Volendo migliorare la soluzione con il metodo di Newton, si ha $1/k = 0.2$ e $(k-1)/k = 0.8$ e

$$x_1 = 2\left(0.8 \cdot 1.14375 + 0.2\frac{1.71875}{1.14375^4}\right) = 2 \cdot 1.11587 = 2.23174$$

$$x_2 = 2\left(0.8 \cdot 1.11587 + 0.2\frac{1.71875}{1.11587^4}\right) = 2 \cdot 1.11441 = 2.22881$$

Vediamo infine una semplice implementazione al calcolatore in linguaggio C: il programma seguente approssima e calcola la radice $k-$esima con il metodo descritto. La funzione `kthroot()` è più lenta rispetto alla `pow()` inclusa in `#math.h`; quest'ultima tuttavia non è universale nel senso che viene implementata in modo diverso dai vari compilatori, e talvolta ottimizzata per sfruttare le particolarità dell'architettura di destinazione.

```
#include <stdio.h>
#include <math.h>

void kthroot(float, int);

int main ()
{
   int k;
   float a, sqa;

   printf ("ordine della radice (intero>0): ");
   scanf("%d", &k);
   printf ("radicando (reale>0): ");
   scanf ("%f", &a);

   printf ("\nradice esatta: %f\n", pow (a, 1.0/k));

   kthroot (a, k);

   return 0;
}

void kthroot(float a, int k)
{
   float r1, r2, fr, rk;
   int ea, sha, i;

   fr = frexp(a, &ea);           // separa mantissa ed esponente
   sha = ea % k;                 // riduzione modulo k dell'esponente
   if (sha > 1)
      sha -= k;                  // (ea - sha) deve essere divisibile per k
   ea = (int)((ea - sha) / k);   // esponente della radice k-esima
   fr = ldexp(fr, sha);          // 1/2^(k-1) <= fr < 2

   /* prima approsssimazione */
   fr = 1.0 + (fr - 1.0) / k;
   r1 = ldexp(fr, ea);
   printf ("\n-prima approssimazione: %f", r1);

   /* iterazioni di Newton-Raphson */
   do {
      r2 = r1;
      rk = r1;
      for (i=2; i<k; ++i)
         rk *= r1;
      r1  = ((k - 1.0) * r1 + (a / rk)) / k;
      printf ("\n-iterazione: %f", r1);
   } while (fabs(r2 - r1) > 1e-6);
}
```