# RADICI QUADRATE E RADICI ENNESIME

Risposta alla domanda comparsa su Quora:
Quanto fa la radice quadrata di 63,8?

(Con estensione alle radici ennesime di quasiasi numero reale, e un cenno agli antilogaritmi, ad uso di chi è veramente curioso.)

Sono anch'io un sostenitore del metodo iterativo indicato da altra risposta. È evidente la ragione per cui funziona: Solo dividendo un numero **d** per la sua radice quadrata **R** si ottiene come quoziente un'identico valore **R**. Scegliamo una prima approssimazione $\mathbf{c_0}$, dividiamo $\mathbf{d/\ c_0}$, troviamo $\mathbf{q_0}$. Se $\mathbf{q_0}$ è minore di $\mathbf{c_0}$, vuol dire che $\mathbf{c_0}$ è maggiore della radice quadrata di **d**, cioè **R**, mentre se si ottiene un risultato $\mathbf{q_0}$ maggiore di $\mathbf{c_0}$, vuol dire che $\mathbf{c_0}$ è minore della radice. In ogni caso R sta tra i due valori, e possiamo prendere come nuova approssimazione la media fra $\mathbf{c_0}$ e $\mathbf{q_0}$, cioè $\mathbf{c_1 = (c_0 + q_0)/2}$, dove $\mathbf{q_0}$ è, appunto, $\mathbf{d/\ c_0}$, e sarà più vicino a **R** di $\mathbf{q_0}$. Questa è la regola riportata su Quora. Ripetendo le stesse operazioni con $\mathbf{c_1}$ invece di $\mathbf{c_0}$ e con le "iterazioni" successive, otterremo valori $\mathbf{q_i}$ e poi $\mathbf{c_i}$, sempre più vicini alla radice **R**: di fatto, quando saremo abbastanza vicini a R guadagneremo almeno due cifre ad ogni iterazione.

Il metodo funziona praticamente sempre, anche partendo da approssimazioni completamente sballate, come per esempio "proporre"200 in prima approssimazione per la radice quadrata di 63.8. Provare per credere (vedi Appendice). Inoltre, esso si presta ad essere applicato scrivendo un semplice programma in qualsiasi banale linguaggio, come, ad esempio SmallBasic. Quest'ultimo è scaricabile gratuitamente in rete e, come sovente accade in Microsoft, oltre ad essere gratuito, ha un sacco di gratuite complicazioni con un alto valore di inutilità.

Dove non concordo con un'eccellente risposta già data, è nella frase: *Purtroppo, calcolare 1/x è comunque scomodo, quindi non è un metodo che si può usare "a mano"!*. Mamma mia, direi io, se si ha paura di fare delle divisioni a mano, e non si è masochisti intellettuali, che si stia lontani dalla matematica! Se si soffre di insonnia, si segua il consiglio di Orazio:

*Ter uncti*
*transnanto Tiberim, somno quibus est opus alto,*
*inriguumque mero sub noctem corpus habento.* (Sermones, II,1,7–10)

"Quelli che soffrono d'insonnia, si ungano bene, e attraversino tre volte il Tevere a nuoto. Inoltre verso sera si bevano un bel po' di vino puro".

Tanto più che ora vorrei mostrare come lo stesso metodo possa essere ricavato con un altro ragionamento, che ha il vantaggio di poter essere esteso a radici ennesime di qualsiasi

grado purché intere, ma sempre al prezzo di fare delle divisioni a mano, se non si ha una piccola calcolatrice che fa le quattro operazioni, e senza usare i prodotti della *Logaritmi & Co.*

Ora, se ci si vogliono sporcare le mani di grasso sotto la macchina dell'iterazione si vede che in pratica abbiamo applicato il seguente algoritmo, valido per qualsiasi radice ennesima (n = intero).

Si voglia la radice **n** del numero **d**

1) si prova con una **prima approssimazione** (qualsiasi! – naturalmente, quanto più lontana è l'approssimazione dal risultato corretto, tante più iterazioni andranno fatte) che chiameremo $\mathbf{c_0}$.

2) si suppone che il risultato corretto sia dato da $c_0+k_0$, cioè $(c_0 + k_0)^n = d$**,** dove $\mathbf{k_0}$ è incognito.

3) Se $k_0$ fosse piccolo – ma noi ci infischiamo di questa limitazione, e agiremo come se lo fosse - potremmo tenere i primi due termini dello sviluppo di $(c_0+ k_0)^n$ ,che sono:

$$d = (c_0 + k_0)^n \cong c_0^n + n\, c_0^{n-1} k_0$$

Da cui:

$$d - c_0^n = n\, c_0^{n-1} k_0$$

Noi vogliamo la correzione incognita $k_0$

$$k_0 = \frac{d - c_0^n}{n\, c_0^{n-1}}$$

La successiva approssimazione sarebbe quindi

$$c_1 = c_0 + k_0 = c_0 + \frac{d - c_0^n}{n\, c_0^{n-1}}$$

Qui si ricomincia da capo, mettendo questa volta $c_1$ al posto di $c_0$. Eseguendo le stesse operazioni con i simboli cambiati si troverà un nuovo $k_1$ e una nuova approssimazione $c_2 = c_1 + k_1$.

La formula iterativa generica sarà quindi:

$$c_{i+1} = c_i + k_i = c_i + \frac{d - c_i^n}{n\, c_i^{n-1}}$$

Anche questo sistema funziona, naturalmente più lentamente (ma , a mio parere, sempre in modo sorprendentemente rapido).

Vediamo che cosa succede nel caso n=2:

Abbiamo:

$$c_1 = c_0 + k_0 = c_0 + \frac{d - c_0^2}{2\,c_0} = c_0 + \frac{d}{2\,c_0} - \frac{c_0}{2} = \frac{1}{2}\left(c_0 + \frac{d}{c_0}\right)$$

cioè l'approssimazione riportata altrove, che poi va iterata ad libitum.

OK, ci sono divisioni da fare, ma qui si ammette che esistano persone che non abbiano paura delle quattro operazioni! Agli altri, che importa conoscere la radice ennesima di un numero?

**APPENDICE:**

**I. (ELEMENTARE) Programma Small Basics: lo si può copiare direttamente su SmallBasic:**

```
TextWindow.WriteLine("Metodo iterativo per calcolare ")
TextWindow.WriteLine("le radici ennesime di un numero reale.")
start:
TextWindow.Write("numero originale?  ")
d = TextWindow.Read()
TextWindow.Write("indice della radice?  ")
n = TextWindow.Read()
TextWindow.Write("prima iterazione ?  ")
c = TextWindow.Read()
i = 0
While i<30
  c= c+(d-Math.Power(c, n))/(n*Math.Power(c, n-1))
  i=i+1
TextWindow.WriteLine("Iteration n."+ i + ": c ("+ i + ") = " + c )
EndWhile
Goto start
```

**NOTA:**

**Il programma elementare da me suggerito è scritto in SmallBasic, linguaggio magari non avanzatissimo, ma che può eseguire potenze con esponenti non interi.**

Esso può quindi risolvere (fra gli altri) un nuovo problema interessante.

Supponiamo di partire dal caso $10^x=a$, dove, evidentemente, x è il logaritmo di a in base 10. Sarà quindi $10 = a^{(1/x)}$ e il programma, noti 10 e 1/x, ci dà la radice "1/x-esima" di 10, cioè a, ovvero l'antilogaritmo decimale di x. E, naturalmente, in luogo di 10 possiamo mettere qualsiasi base, inclusa e. Provare per credere.

Ma in fondo abbiamo barato, perché il termine $d/(c(i)^{(1/x-1)})$ presente nella formula iterativa richiede un'operazione che si può senz'altro fare, ma richiede un algoritmo abbastanza lungo, che toglie la bellezza al metodo. Però, se lasciamo che ci pensi il nostro calcolatore….

## II. RISULTATI

Metodo iterativo per calcolare le radici ennesime di un numero reale.

**ESEMPIO 1: RADICE SETTIMA DI 63.8**

numero originale? 63.8

***indice della radice? 7***

prima approssimazione ? 1

Iteration n.1: c (1) = 9,971428571428571428571428571
Iteration n.2: c (2) = 8,546948047615246343901942633
Iteration n.3: c (3) = 7,325978850016909383588985922
Iteration n.4: c (4) = 6,279469398961965876511257 5613
Iteration n.5: c (5) = 5,382550998825081989731532 1715
Iteration n.6: c (6) = 4,613989936789566510688782 4649
Iteration n.7: c (7) = 3,955793149525168302083489 3521
Iteration n.8: c (8) = 3,393058441244935989637601 3098
Iteration n.9: c (9) = 2,914308560389624429354954 4101
Iteration n.10: c (10) = 2,512855563386526695213079 8428
Iteration n.11: c (11) = 2,190076939825173317551856 2597
Iteration n.12: c (12) = 1,959806210845245930331749 7536
Iteration n.13: c (13) = 1,840692338018810145831177 8880
Iteration n.14: c (14) = 1,812070346702030069234275 1628
Iteration n.15: c (15) = 1,810640955718515336358555 5647
Iteration n.16: c (16) = 1,810637561547882904414527 8000
Iteration n.17: c (17) = 1,810637561528794584292157 3342
**Risultato esatto (da Google): 1.81063756153**

**ESEMPIO 2: RADICE QUINTA DI 987654321**

numero originale? 987654321

***indice della radice? 5***

prima approssimazione? 50

Iteration n.1: c (1) = 71,604938272
Iteration n.2: c (2) = 64,797810031254767471989146660
Iteration n.3: c (3) = 63,042778202264480238291291801
Iteration n.4: c (4) = 62,939507429117539328345859040
Iteration n.5: c (5) = 62,939167423491041905442842182

Iteration n.6: c (6) = 62,939167419817472780337273372
**Radice correttta (da Google): 62.9391674198**

**ESEMPIO 3: RADICE DICIASSETTESIMA DI 89723459871317854**

numero originale? 89723459871317854

indice della radice? 17

prima approssimazione? 10

Iteration n.1: c (1) = 9,93954976394892855294117647 1
Iteration n.2: c (2) = 9,93642389068296372000762528 9
Iteration n.3: c (3) = 9,93641599901232012821308570 1
Iteration n.4: c (4) = 9,93641599896217909586044715 6
Iteration n.5: c (5) = 9,93641599896217749331653431 8
Iteration n.6: c (6) = 9,93641599896217784509446640 4
Iteration n.7: c (7) = 9,93641599896217819687239849 0
Iteration n.8: c (8) = 9,93641599896217659432848565 2
Iteration n.9: c (9) = 9,93641599896217694610641773 8
Iteration n.10: c (10) = 9,93641599896217729788434982 4
**Risultato corretto (da Google)=** 9.93641599896

Ho notato che Small Basic non è attendibile oltre la 15^a cifra. Lascio all'eventuale industrioso lettore il compito di esplorare queste anomalie.

Incidentalmente, la radice quadrata di 63.8, con la balorda prima approssimazione c0 = 200, dà i seguenti risultati

numero originale? 63.8

indice della radice? 2

Prima approssimazione ? 200

Iteration n.1: c (1) = 100,1595
Iteration n.2: c (2) = 50,39824200525162366026188229 8
Iteration n.3: c (3) = 25,83207958869464650512203875 4
Iteration n.4: c (4) = 14,15093843618801776333276211 5
Iteration n.5: c (5) = 9,32973667490257179866074563 4
Iteration n.6: c (6) = 8,08404308070127372966856451 5
Iteration n.7: c (7) = 7,98806681516536644644006161 3
Iteration n.8: c (8) = 7,98749023989632187326347156 5
Iteration n.9: c (9) = 7,98749021908634***33934453129 93***

E di qui il risultato non sembra spostarsi più, mentre il risultato corretto sembra essere quello che si ottiene iniziando con la prima approssimazione 8:
Iteration n.3: c (3) = 7,98749021908634***1150723796149***

In prima battuta direi che l'errore proviene dalla divisione d/(c(i)), la cui precisione non può superare un dato numero di cifre.