Another Method for Extracting Cube Roots

Brian J. Shelburne
Dept of Math and Computer Science
Wittenberg University

Probably the easiest way (?) to compute the cube root of a is to use the iteration formula $x_{n+1} = \frac{2x_n^3 + a}{3x_n^2}$. However, while checking out an article in the December 1958 issue of the *Communications of the ACM* I ran across a paper that demonstrated another method for extracting cube roots. The method was useful (?) since it was done using fixed point arithmetic, it required no division and could be modified so only minimal multiplication was needed (both "expensive" operations for computers in the 1950's). It's similar to the method of extracting a square root by hand.

The Theory

The theory behind this technique is simple. Let X be the number whose cube root we want. Let the number we want, $\sqrt[3]{X}$ be represented by the sum $\sum_{i=1}^{\infty} d_i 10^{n-i}$ where d_i is a digit between 0 and 9 and n is the number of digits above the decimal point. Letting $a_i = d_i 10^{n-i}$ we can write $X = (a_1 + a_2 + ... + a_n + ...)^3$. We will show how to compute each term a_k or more exactly each digit d_i in an iterative fashion.

1. Observe the following

For positive integer k, $(a_1 + a_2 + ... + a_k)^3$ approximates X and so $(a_1 + a_2 + ... + a_k)$ approximates $\sqrt[3]{X}$. Using the binomial theorem we can expand $(a_1 + a_2 + ... + a_{k-1} + a_k)^3$ in terms of $(a_1 + a_2 + ... + a_{k-1})$ and a_k, specifically

$$(a_1 + a_2 + ... + a_{k-1} + a_k)^3 = (a_1 + a_2 + ... + a_{k-1})^3 + \left[3 \cdot (a_1 + a_2 + ... + a_{k-1})^2 \cdot a_k + 3 \cdot (a_1 + a_2 + ... + a_{k-1}) \cdot a_k^2 + a_k^3\right]$$

For example

$$(a_1 + a_2)^3 = a_1^3 + \left[3 \cdot a_1^2 a_2 + 3 \cdot a_1 a_2^2 + a_2^3\right].$$

Continuing we have

$$(a_1 + a_2 + a_3)^3 = (a_1 + a_2)^3 + \left[3 \cdot (a_1 + a_2)^2 a_3 + 3 \cdot (a_1 + a_2) a_3^2 + a_3^3\right] \text{ etc.}$$

The idea is to work by stages so that given $(a_1 + a_2 + ... + a_{k-1})$ at one stage we choose a_k so that $(a_1 + a_2 + ... + a_{k-1} + a_k)^3$ is a better approximation to X.

2. Exactly how do we use $(a_1 + a_2 + \ldots + a_{k-1})$ to obtain $(a_1 + a_2 + \ldots + a_{k-1} + a_k)$?

Begin by finding the *largest* value of a_1 such that the *error* $X_1 = X - a_1^3$ is non-negative, or equivalently finding the largest value of a_1 such that a_1^3 is less than or equal to X.

Knowing the value of a_1 find the *largest* value of a_2 such that the *error* $X_2 = X_1 - \left[3 \cdot a_1^2 a_2 + 3 \cdot a_1 a_2^2 + a_2^3\right]$ is non-negative or equivalently such that $\left[3 \cdot a_1^2 a_2 + 3 \cdot a_1 a_2^2 + a_2^3\right]$ is less than or equal to X_1. Note that $X_2 = X - (a_1 + a_2)^3$

In general given $(a_1 + a_2 + \ldots + a_{k-1})$ find the *largest* value of a_k such that the error $X_k = X_{k-1} - \left[3 \cdot (a_1 + a_2 + \ldots + a_{k-1})^2 a_k + 3 \cdot (a_1 + a_2 + \ldots + a_{k-1}) a_k^2 + a_k^3\right]$ is non-negative. Note that $X_k = X - (a_1 + a_2 + \ldots + a_k)^3$ [1].

3. A Notational Refinement!

Before going further, let's introduce some notation to make our formulas above less cumbersome and, as we will see, to simplify our calculations.

Let

$$u_k = (a_1 + a_2 + \ldots + a_{k-1})$$

and let

$$v_k = u_k^2 = (a_1 + a_2 + \ldots + a_{k-1})^2$$

So given u_k and $v_k = u_k^2$ find the *largest* value of a_k such that the error $X_k = X_{k-1} - \left[3 \cdot v_k \cdot a_k + 3 \cdot u_k \cdot a_k^2 + a_k^3\right]$ is non-negative! Again note that $X_k = X - u_{k+1}^3$

An Example

Unfortunately it's not clear how to apply the above theory in a *practical* and *efficient* way to actually compute a cube root; that's what the following example will show.

For example compute $\sqrt[3]{79201}$.

We begin by partitioning the digits of 79201 into groups of three (i.e. 79 201.000) starting at the decimal point. At each stage we will bring down for consideration the next group of three digits. Essentially we are scaling our calculations by 1000, the cube of 10.

[1] Observe that $X - (a_1 + a_2 + \ldots + a_k)^3 \geq 0$ so $(a_1 + a_2 + \ldots + a_k)$ is the largest k digit approximation less than or equal to the cube root of X.

We begin by only working with the left-most group of digits, essentially treating $\sqrt[3]{79201}$ like $\sqrt[3]{79.201}\times 10$. This allows us to "work" with single digit values between 0 and 9; that is each a_k we find will be a digit d_k between 0 and 9.

Stage 1. Find the largest digit d_1 such that $d_1^3 \le 79$.

This is 4 (it helps to know the cubes of the first nine integers: 1, 8, 27, 64, 125, 216, 343, 512, 729). Subtract 64 from 79 and bring down the next three digits.

```
   79 201
-  64
   ----------
   15 201   = X1
```

Stage 2. We need to find the largest a_2 such that the error $X_1 - \left[3\cdot a_1^2 a_2 + 3\cdot a_1 a_2^2 + a_2^3\right]$ is non-negative. Because bringing down the next three digits scales X_1 by 1000, the current cube root approximation 4 needs to be scaled by 10, i.e. $a_1 = 4\times 10$. We need to compute $3\cdot a_1 = 3\cdot(4\times 10) = 120$ and $3\cdot a_1^2 = 3\cdot(4\times 10)^2 = 4800$. a_2 will be a digit between 0 and 9

In finding the largest digit d_2 such that $4800\times d_2 + 120\times d_2^2 + d_2^3$ is less than 15201, 3 seems like a good estimate for d_2 but it's too high. So d_2 equals 2. $4800\times 2 + 120\times 2^2 + 2^3 = 10088$ which when subtracted from 15201 results in 5113.

```
   15 201 000
-  10 088
   ----------
    5 113 000 = X2
```

Note that 42 is the largest integer less than or equal to $\sqrt[3]{79201}$. Shift in three more digits and repeat with X_2 = 5 113 000.

Stage 3. Because of the way we scale our values $(a_1 + a_2) = u_3 = (4\times 100 + 2\times 10) = 420$. We need to compute $3\cdot(a_1 + a_2) = 3\cdot(420) = 1260$ and $3\cdot(a_1 + a_2)^2 = 3\cdot v_3 = 3\cdot(420)^2 = 529,200$ and find the largest digit d_3 such that the error $X_2 - 529,200\times d_3 + 1260\times d_3^2 + d_3^3$ is non-negative. We estimate $d_3 = 9$ which works since $529,200\times 9 + 1260\times 9^2 + 9^3 = 4,865,589$ so subtracting

```
    5 113 000 000
-   4 865 589
    ----------
      247 411 000 = X3
```

With $d_3 = 9$ our approximation to $\sqrt[3]{79201}$ is 42.9. Shift in three more digits and repeat with $X_3 =$ 247,411,000

The TANSTAAFL[2] Effect

The obvious draw backs to this method are 1) the computational complexity of working with large integer values, 2) the need to compute the square of a large integer, i.e. $(a_1+a_2+...+a_{k-1})^2$ and, 3) the need to find the largest digit d_k such that the error at the k[th] stage $X_k - X_{k-1} - \left[3\cdot(a_1+a_2+...+a_{k-1})^2 a_k + 3\cdot(a_1+a_2+...+a_{k-1})a_k^2 + a_k^3\right]$ is non-negative, usually by trial and error.

Reducing Computational Complexity - I

We can simplify the squaring of $(a_1+a_2+...+a_{k-1})^2$ at the k[th] step by using previously computed values. Recall that u_k is the value of $(a_1+a_2+...+a_{k-1})$ and $v_k = u_k^2$ is the value of $(a_1+a_2+...+a_{k-1})^2$ at the k[th] step. It's not difficult to derive a pair of interlocking formulas to compute u_k and v_k.

$$u_k = \begin{cases} 0 \; if \; k = 0 \\ (u_{k-1} + d_{k-1})\cdot 10 \; if \; k > 0 \end{cases}$$

and

$$v_k = \begin{cases} 0 \; if \; k = 0 \\ (v_{k-1} + 2\cdot u_{k-1}\cdot d_{k-1} + d_{k-1}^2)\cdot 100 \; if \; k > 0 \end{cases}$$

Given u_{k-1} and v_{k-1} the computational cost of computing $v_k = (a_1+a_2+...+a_{k-1})^2$ is reduced to an addition followed by a multiplication by 10 to obtain u_k plus *two* additions and *three* simple multiplications (by 2, by the digit d_k and by 100) to obtain v_k (assuming a simple table lookup to find the square of the digit d_k).[3]

Thus we can simplify our calculations if at each step we also calculate and use u_k and v_k as seen below where the 4[th] digit for $\sqrt[3]{79201} \approx 42.94$ obtained.

[2] There Ain't No Such Thing As A Free Lunch – from *The Moon is A Harsh Mistress* by Robert Heinlein

[3] The article in CACM mentioned that this method had been successfully and efficiently implemented on an IBM 650 – a decimal computer where multiplications by powers of ten were easily implemented by left shifts. Even on a binary machine multiplication by ten can be efficiently implemented by a left shift by 3 bits added to a left shift by 1 bit – essentially x*8 + x*2. The IBM 650 also had a table loop-up operation.

k	d_k	u_k	v_k	$3v_k^2d_k+3u_kd_k^2+d_k^3$	X_k
0	0	0	0	0	79.201
1	4	0	0	64	15.201
2	2	40	1600	10088	5113
3	9	420	176400	4865589	247411
4	4	4290	18404100	221055184	26355816

You can easily check that $v_3 = (v_2 + 2 \cdot u_2 \cdot d_2 + d_2^2) \cdot 100 = (1600 + 2 \cdot 40 \cdot 2 + 2^2) \cdot 100 = 176400$

Reducing Computational Complexity – II

The *difference* between adjacent terms of the form $3 \cdot v_k \cdot m + 3 \cdot u_k \cdot m^2 + m^3$ for m = 1, 2, etc. can be written as

$$3 \cdot v_k \cdot m + 3 \cdot u_k \cdot m^2 + m^3 - \left(3 \cdot v_k \cdot (m-1) + 3 \cdot u_k \cdot (m-1)^2 + (m-1)^3\right) = 3 \cdot v_k + 3 \cdot u_k \cdot (2m-1) + \Delta m^3$$

where $\Delta m^3 = m^3 - (m-1)^3$ *denotes the difference* between adjacent cubes and $(2m-1)$ *is* the difference between adjacent squares.

The computational cost of estimating the largest d_k such that $X_{k-1} - 3 \cdot v_k \cdot d_k + 3 \cdot u_k \cdot d_k^2 + d_k^3$ is non-negative can be simplified by subtracting $3 \cdot v_k + 3 \cdot u_k \cdot 1 + 1$ from X_{k-1} and continuing to subtract expressions of the form $3 \cdot v_k + 3 \cdot u_k \cdot (2m-1) + \Delta m^3$ until a negative value is obtained (then restoring the previous value). The difference between adjacent cubes (i.e. 1, 7, 19, 37, 61, 91, 127, 169, 217, 271) is easily found by table look up.

For example, the calculation for k = 4 above (where $3 \cdot v_4 = 55,212,300$ and $3 \cdot u_4 = 12,870$) can be redone using the following table.

d_4	$55212300+12870\times(2k-1)+\Delta d_4^3$	$X_3\times1000 = 247411000$
1	55212300+12870×1+1 = 55225171	192185829
2	55212300+12870×3+7 = 55250917	136934912
3	55212300+12870×5+19 = 55276669	81658243
4	55212300+12870×7+37 = 55302427	26355816
5	55212300+12870×9+61 = 55328191	-28972375
6	55212399+12870×11+127	…

Computationally for each row you need only one multiplication (by an odd integer), two additions and one table look up for the first difference of the cube.

Computing a Cube Root by Hand

Finding the cube root by hand can be done in a way similar to the method used for extracting square roots. For example, suppose we want to find $\sqrt[3]{23}$. We start with a division like structure.

```
--------------
| 23.000 000
```

Begin by finding the largest integer cube less than or equal to 23 (8) and putting its cube root above where the quotient goes, subtracting the cube from the "dividend" and bringing down the next three zeros.

```
   2
--------------
| 23.000 000
-  8
  ------
  15 000
```

Now find the largest digit d_2 such that $3\cdot 2^2\cdot 100\cdot d_2 + 3\cdot 2\cdot 10\cdot d_2^2 + d_2^3 = 1200\cdot d_2 + 60\cdot d_2^2 + d_2^3$ is less than or equal to 15,000. Since the first term (1200) dominates this expression, we estimate 8 for d_2.

To simply the calculation of $3\cdot 2^2\cdot 100\cdot d_2 + 3\cdot 2\cdot 10\cdot d_2^2 + d_2^3 = 1200\cdot d_2 + 60\cdot d_2^2 + d_2^3$ we're going to factor out the d_2 and express the remaining factor as a sum of three terms: 1200, $60\cdot d_2$ and d_2^2 (We express 1200 below as 3 × 400 for reasons we'll explain at the next step.). We then multiply this sum by 8 and subtract from our "dividend" and bring down the next three digits

```
3×400 =   1200
3×20×d2=   480
     d2²=   64
       -------
          1744
```

```
     2.  8
--------------
| 23.000 000
-  8
  ------
  15 000
- 13 952
  ------
    1 048 000
```

Alternately – take the current "quotient" times 10, square it and multiply by 3. Then again take the quotient times 10 multiply by 3 and by d_2 (i.e. 8). Finally square d_2 and add all three terms together.

At the next step complexity becomes apparent in the calculation of the expression $3 \cdot v_k \cdot d_k + 3 \cdot u_k d_k^2 + d_k^3$ or more precisely $3 \cdot v_k + 3 \cdot u_k \cdot d_k + d_k^2$ where $u_k = (a_1 + a_2 + ... + a_{k-1})$ and $v_k = u_k^2$. So for the next round we use the recursive formula for v_k to obtain $v_3 = (400 + 2 \cdot 20 \cdot 8 + 64) \cdot 100 = 78400$. Since $3 \cdot v_k = 235200$ dominates the expression above we estimate 4 as the next digit

```
                                    2.   8    4
                                 --------------
                                 | 23.000 000
                                  - 8
                                    ------
3×78400 = 235200                    15 000
3×280×d3 =   3360                 - 13 952
      d3² =     16                  ------
          ------                     1 048 000
          238576                     - 954 304
                                     ---------
                                        93 696
```

Observe that once the value of $3 \cdot v_k$ is obtained, it is fairly easy to estimate a value for the digit d_k - and the rest is just so much routine calculation.

An Afterword - So What Good is This?

Given the ubiquity of cheap powerful calculators that can easily do division, why use this technique? There is no good reason *but* it is interesting that with this technique you *could* calculate a cube root by hand[4] or program a very unsophisticated computer to do so using only minimal multiplication and no division.

The technique to obtain a cube root is an extension of the standard technique to obtain a square root. In this case given $X = (a_1 + a_2 + ... + a_n + ..)^2$ observe that $(a_1 + a_2 + ... + a_k)^2$ can be expanded as $(a_1 + a_2 + ... + a_{k-1} + a_k)^2 = (a_1 + a_2 + ... + a_{k-1})^2 + [2 \cdot (a_1 + a_2 + ... + a_{k-1}) \cdot a_k + a_k^2]$. Applying the technique discussed above to square roots (except we group digits by twos), at the kth step we find the largest digit d_k such that the error at the kth stage $X_k = X_{k-1} - 2 \cdot u_k \cdot d_k + d_k^2$ where $u_k = (a_1 + a_2 + ... + a_{k-1})$ is non-negative. This is usually expressed as take the current "quotient", double and multiply by 10 and using this trial divisor and divide into the dividend to estimate the next digit which is then added to the trial divisor i.e. $2 \cdot u_k + d_k$. Now multiply the

[4] Isaac Asimov's short story "The Feeling of Power" tells the tale where all calculations are done my hand-held calculators and people have forgotten how to do calculations by hand – until one day someone rediscovers hand calculations. The story has a tragic ending as the military immediately sees certain "applications" for hand calculation!

digit by the augmented trial divisor which gives you $2 \cdot u_k \cdot d_k + d_k^2$) and subtract from the dividend (the result should be positive). Repeat.

Example: Find $\sqrt{23}$

Partition the digits into groups of two and begin by finding the largest digit whose square less than or equal to the leading group. Put the digit on top, subtract the square from the dividend and bring down the next two digits.

```
      4
    -------
    | 23.00 00
4   - 16
      --
      7 00
```

Take the quotient (4) double and multiply by 10 (80) and use this as trial divisor. Put the digit result on top, add the digit to the trial divisor, multiply the trial divisor by the digit and subtract.

```
      4  7
    -------
    | 23.00 00
 4  - 16
      --
      7 00
87    - 6 09
      ----
        91 00
```

Repeat: Double 47 and multiply by 10 to obtain 940 as a trial divisor. 940 goes into 9100 approximately 9 times.

```
      4  7  9
    ----------
    | 23.00 00
 4  - 16
      --
       7 00
87    -  6 09
       ----
         91 00
949   -    85 41
         -----
           5 59 00
```

Thus 4.79 approximates $\sqrt{23}$.

The basic theory given above can also be applied to roots higher than the cube, e.g. fourth and fifth etc. For the 4th root case observe

$$(a_1 + a_2 + ... + a_k)^4 = (a_1 + a_2 + ... + a_{k-1})^4 + [4 \cdot w_k \cdot a_k + 6 \cdot v_k \cdot a_k^2 + 4 \cdot u_k \cdot a_k^3 + a_k^4]$$

Where u_k and v_k are defined as above and $w_k = u_k^3$! The same method can be applied to find the 4th root (group digits by 4) but the computational complexity problems are even more severe!

References

Sugai, Iwao; “Extraction of Roots by Repeated Subtractions for Digital Computers”; *Comm. of A.C.M.*; Vol 1 No 12; December 1958; pp 6 – 8.

Asimov, Isaac; "The Feeling of Power"; *The Mathematical Magpie*, Clifton Fadiman, ed.; Simon and Schuster; New York; 1962.