

10/06/98

EXTENDING THE BABYLONIAN ALGORITHM

Mathematics and Computer Education, Vol. 33, No. 2, 1999, pp. 120-128.

Thomas J. Osler
Department of Mathematics
Rowan University
Glassboro, NJ 08028

osler@rowan.edu

1. Introduction

The Babylonians devised a remarkable iterative algorithm [2] for the calculation of square roots in about 1500 BC. This algorithm is easy to understand and remarkably fast, yielding about 26 decimal digits accuracy in only five iterations. The algorithm has appeared in introductory computer programming texts as well as precalculus, calculus (see [1], pages 4-5) and numerical analysis texts.

In this paper we begin by reviewing the Babylonian square root algorithm. We then examine how the algorithm might be extended to the calculation of cube roots, fourth roots, etc. After guessing at a new algorithm, we give a simple BASIC program that allows us to examine numerically how quickly the new algorithm converges. We try modifying the algorithm to improve its speed and discover a simple method for this purpose. Finally we examine the algorithm mathematically and see the reason behind our experimental discoveries. A different modification of the Babylonian algorithm has appeared in [3].

2. The Babylonian algorithm

We begin by reviewing the Babylonian algorithm for finding the square root of the number A . We start by making a guess for \sqrt{A} , and call it x_0 . Figure 2.1 shows the case where $x_0 < \sqrt{A}$.

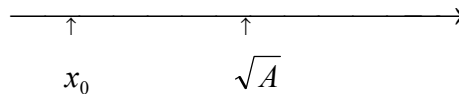


Figure 2.1

Since x_0 is less than \sqrt{A} , then $1 < \sqrt{A}/x_0$, and thus $A/x_0 = (\sqrt{A}/x_0)\sqrt{A}$ will be greater than \sqrt{A} as shown below in Figure 2.2.

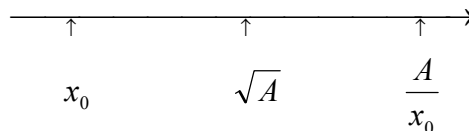


Figure 2.2

Since the \sqrt{A} is between two values, it seems reasonable to try for our next approximation the average of these two values. Thus we try

$$x_1 = \frac{1}{2} \left(x_0 + \frac{A}{x_0} \right)$$

as our second approximation to \sqrt{A} as shown in Figure 2.3.

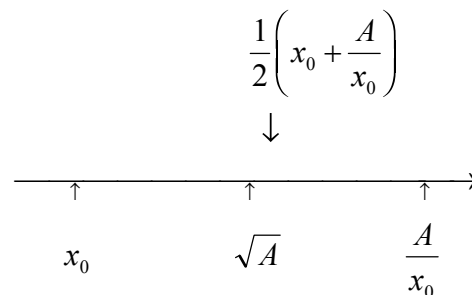


Figure 2.3

Continuing in this way we take for our third approximation $x_2 = \frac{1}{2} \left(x_1 + \frac{A}{x_1} \right)$, and we

now have the general recursion relation

$$(2.1) \quad x_{n+1} = \frac{1}{2} \left(x_n + \frac{A}{x_n} \right) \quad \text{for } n = 0, 1, 2, 3, \dots$$

This last relation is called the Babylonian algorithm for the \sqrt{A} . In the next section we will show how rapidly the sequence $\{x_n\}$ converges to \sqrt{A} by a numerical example.

3. Numerical calculations with the Babylonian algorithm

It is easy to examine our Babylonian algorithm (2.1) numerically using a calculator or a computer. Program 3.1 below is written in QBASIC and nicely demonstrates the algorithm. In line 120 we ask for double precision arithmetic to give us 16 digits of accuracy. In line 130 we ask for the square root of two ($A = 2$) and start with iterations with $x_0 = 1$ ($X = 1$). In line 140 we calculate the next value of our iteration and call it XX. Lines 150 to 170 set up an iteration counter (N), print the iteration, and complete the iteration loop.

Program 3.1

```

100 'Babylonian Algorithm
110 'Find square root of A
120 DEFDBL A-Z
130 A = 2: X = 1
140 XX = .5 * (X + A / X)
150 N = N + 1
160 PRINT N; XX
170 IF XX <> X THEN X = XX: GOTO 140

```

Examination of the Program Output in Table 3.1 shows that over 14 correct decimal digits are calculated in only five iterations. In fact the number of correct decimal digits, roughly speaking, *doubles* each iteration. This is known as “quadratic convergence”. In only 21 iterations, over one million decimal digits would be computed accurately!

Table 3.1

n	x_n	correct digits
1	1.5	0
2	1.416666666666667	2
3	1.41421568627451	5
4	1.41421356237469	11
5	1.414213562373095	14+
6	1.414213562373095	

4. Extending the algorithm to cube roots

Can we modify the Babylonian algorithm to give us cube roots? Suppose we want to find $\sqrt[3]{A}$. Reasoning as before we start with a close guess x_0 . Suppose

$x_0 < \sqrt[3]{A}$, then $1 < \frac{\sqrt[3]{A}}{x_0}$ and therefore $\sqrt[3]{A} < \frac{\sqrt[3]{A}}{x_0} \frac{\sqrt[3]{A}}{x_0} \sqrt[3]{A} = \frac{A}{x_0^2}$. These numbers are

shown in Figure 4.1.

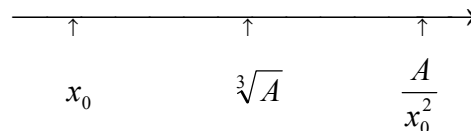


Figure 4.1

Once again we have two values on either side of the desired number, so why not take their average. Continuing in this way we get the recursion relation

$$(4.1) \quad x_{n+1} = \frac{1}{2} \left(x_n + \frac{A}{x_n^2} \right) \quad \text{for } n = 0, 1, 2, 3, \dots$$

We can test this iteration by making a slight modification of Program 3.1. Simply change line 140 to: `XX = .5 * (X + A / (X*X))`. Running the program produces the following :

Table 4.1

Program Output for $\sqrt[3]{A}$

n	x_n
1	1.5
2	1.1944444444444444
3	1.298141638122709
4	1.24248215661613
5	1.269009360346939
...	
47	1.259921049894875
48	1.259921049894872
49	1.259921049894874
50	1.259921049894873
51	1.259921049894873

Here the output for the cube root is very different from that for the square root. Only 5 iterations gave us over 15 correct digits of $\sqrt{2}$, but 50 iterations are required for the same accuracy with $\sqrt[3]{2}$! This convergence is not quadratic. A close look at the output reveals that it takes roughly three iterations to get one more digit of decimal accuracy.

Our cube root algorithm is much slower than we might have expected. In the next section we discover a way to improve the speed of convergence.

5. Improving the cube root algorithm.

The square root algorithm produced rapid “quadratic” convergence because the exact \sqrt{A} was very nearly midway between the values x_n and $\frac{A}{x_n}$ as shown in

Figure 2.2. In the cube root case, this midway feature is no longer true. Figure 5.1

shows that the exact $\sqrt[3]{A}$ is closer to the left end x_0 .

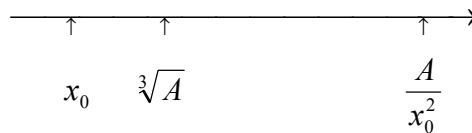


Figure 5.1

Thus we need a weighted average of the values, with the left value weighted higher than the right value. The appropriate iteration could be written as

$$(5.1) \quad x_{n+1} = \alpha x_n + \beta \frac{A}{x_n^2} \quad \text{where } \alpha + \beta = 1.$$

(Here $0 < \alpha < 1$ and $0 < \beta < 1$.) By experimenting with various values of the weights α and $\beta = 1 - \alpha$, we can try to speed up the convergence. Program 5.1 (a modification of Program 3.1), is designed to help us perform this exploration.

Program 5.1

```

100 'Babylonian Algorithm Extended
110 'Find cube root of A
120 DEFDBL A-Z
125 INPUT "Enter Alpha "; ALPHA: BETA = 1 - ALPHA
130 A = 2: X = 1
140 XX = ALPHA * X + BETA * A / (X * X)
150 N = N + 1
160 PRINT N; XX
170 IF XX <> X THEN X = XX: GOTO 140

```

In this modified program line 125 has been added to allow the user to INPUT the value of α . Line 140 has been changed to reflect the recurrence relation (5.1). The following Table 5.1 shows that when $\alpha = 2/3$ we once again get quadratic convergence and only five iterations are required to get 15 decimal digits accuracy. Thus the best Babylonian type algorithm for the cube root of A is given by

$$(5.2) \quad x_{n+1} = \frac{2}{3}x_n + \frac{A}{3x_n^2} .$$

Table 5.1

ALPHA	No. of Iterations for 15 Decimal Digits	
0.3	failure	
0.4	152	
0.5	50	
0.6	22	
0.65	12	
0.66666...	5	(quadratic convergence)
0.67	9	
0.7	16	
0.8	36	

6. Extending the algorithm to the fourth and higher roots.

It is now an easy matter to extend the previous iterative algorithm to the calculation of the p th root of A . We start with a guess x_0 and reason that if this number is to the left of $\sqrt[p]{A}$, then the number $\frac{A}{x_0^{p-1}}$ is to the right and the weighted average should give the next iteration. We conclude that

$$(6.1) \quad x_{n+1} = \alpha x_n + \beta \frac{A}{x_n^{p-1}} \quad \text{with} \quad n = 0, 1, 2, 3, \dots$$

is a promising recursion relation that should converge to $\sqrt[p]{A}$. Program 5.1 can easily be modified to test (6.1). We need only replace line 140 by

$$140 \quad XX = \text{ALPHA} * X + \text{BETA} * A / (X * X * X)$$

for 4 th roots, and add an extra X in the parentheses for each higher root.

The reader can experiment and find that quadratic convergence is obtained for the p th root of A when we take $\alpha = \frac{p-1}{p}$ and $\beta = \frac{1}{p}$. This is an interesting exploration for students. We conclude that the best Babylonian type algorithm for the calculation of $\sqrt[p]{A}$ is

$$(6.2) \quad x_{n+1} = \frac{p-1}{p} x_n + \frac{A}{p x_n^{p-1}} \quad \text{with } n = 0, 1, 2, 3, \dots$$

It is interesting to note that (6.2) is the same as the iteration obtained from the very popular Newton's method, (see [1], pages 363-368). We now conclude our investigation by studying the rate of convergence of the iterations given by (6.1) analytically. This study will reveal why all the experimental features observed above occurred.

7. Mathematical analysis of the speed of the algorithms

Now we will use the power of mathematical analysis to derive an expression of the form

$$\epsilon_{n+1} = a_1 \epsilon_n + a_2 \epsilon_n^2 + a_3 \epsilon_n^3 + \dots,$$

which explains how the error at step $n+1$ of our iterations (ϵ_{n+1}) can be calculated from the error at the previous iteration step (ϵ_n). When we obtain this expression we will explain all the features observed experimentally in previous sections. Notice that the errors should be small, and thus higher powers of ϵ_n diminish rapidly and will be

ignored. Also, if $|a_1| < 1$, then the errors will decrease as the iterations increase, otherwise the errors will not decrease.

Consider the iterations defined by

$$(7.1) \quad x_{n+1} = \alpha x_n + \beta \frac{A}{x_n^{p-1}} \quad \text{with} \quad \alpha + \beta = 1.$$

This is the sequence which we expect to converge to $\sqrt[p]{A}$. We write

$$(7.2) \quad \sqrt[p]{A} = x_n + \epsilon_n$$

where ϵ_n is the error in the n th iteration x_n . The $n+1$ st error is given by

$$\epsilon_{n+1} = \sqrt[p]{A} - x_{n+1}.$$

Replacing x_{n+1} by (7.1) in the above expression we get

$$\epsilon_{n+1} = \sqrt[p]{A} - \alpha x_n - \beta \frac{A}{x_n^{p-1}}.$$

Next using (7.2) we substitute $(\sqrt[p]{A} - \epsilon_n)$ for x_n in the above to get

$$(7.3) \quad \epsilon_{n+1} = \sqrt[p]{A} - \alpha(\sqrt[p]{A} - \epsilon_n) - \beta A(\sqrt[p]{A} - \epsilon_n)^{1-p}.$$

Expanding $(\sqrt[p]{A} - \epsilon_n)^{1-p}$ with the binomial theorem we have

$$(7.4) \quad \begin{aligned} (\sqrt[p]{A} - \epsilon_n)^{1-p} &= A^{\frac{1}{p}-1} \left(1 - \frac{\epsilon_n}{\sqrt[p]{A}} \right)^{1-p} \\ &= A^{\frac{1}{p}-1} \left(1 - \frac{1-p}{1} \left(\frac{\epsilon_n}{\sqrt[p]{A}} \right) + \frac{(1-p)(-p)}{1 \cdot 2} \left(\frac{\epsilon_n}{\sqrt[p]{A}} \right)^2 + \dots \right). \end{aligned}$$

Substituting (7.4) for the last factor in (7.3) we get

$$\epsilon_{n+1} = \sqrt[p]{A} - \alpha(\sqrt[p]{A} - \epsilon_n) - \beta \sqrt[p]{A} \left(1 + \frac{p-1}{1} \left(\frac{\epsilon_n}{\sqrt[p]{A}} \right) + \frac{p(p-1)}{2} \left(\frac{\epsilon_n}{\sqrt[p]{A}} \right)^2 + \dots \right)$$

Rewriting this expression as a power series in ϵ_n we get

$$\epsilon_{n+1} = \sqrt[p]{A}(1 - \alpha - \beta) + (\alpha - (p-1)\beta) \epsilon_n - \frac{\beta p(p-1)}{2\sqrt[p]{A}} \epsilon_n^2 + \dots$$

Since $\alpha + \beta = 1$ the first term on the right vanishes and we have

$$(7.5) \quad \epsilon_{n+1} = (\alpha - (p-1)\beta) \epsilon_n - \frac{\beta p(p-1)}{2\sqrt[p]{A}} \epsilon_n^2 + \dots$$

This completes our mathematical derivation. We will use (7.5) in the next section to explain our previous numerical results.

8. Explanation of numerical results.

We will now show how relation (7.5) can predict all the important features which we observed previously in our numerical experiments.

Case 1: The original Babylonian algorithm

In Table 3.1 we used the recursion relation (6.1) with the initial value $x_0 = 1$, $p=2$, $\alpha = \beta = 0.5$, and $A = 2$. The error relation (7.5) now reads

$$(8.1) \quad \epsilon_{n+1} = 0 \epsilon_n - \frac{1}{2\sqrt{2}} \epsilon_n^2 + \dots \approx -0.354 \epsilon_n^2 .$$

Since our initial guess was $x_0 = 1$, we calculate $\epsilon_0 = \sqrt{2} - x_0 = \sqrt{2} - 1 \approx 0.414$. We can now use (8.1) to find the errors shown below:

Table 8.1

$\epsilon_1 \approx -0.0607$
$\epsilon_2 \approx -0.00130$
$\epsilon_3 \approx -0.000000601$
$\epsilon_4 \approx -0.000000000000128$
$\epsilon_5 \approx -5.80 * 10^{-27}$

This list of errors shows at once why the Babylonian algorithm converges

“quadratically”. You can see the number of zeroes in the errors roughly doubling each

step. Relation (8.1) is dominated by the ϵ_n^2 term which causes this very rapid convergence. If the ϵ_n term had not vanished in (8.1) the convergence would have been much slower. This Table 8.1 explains Table 3.1 nicely.

Case 2: Modified algorithm for cube roots

In Table 4.1 we see the results of using the recursion (6.1) with $p=3$ (cube roots), $A=2$ and $\alpha = \beta = 0.5$. Since the initial guess $x_0 = 1$, we calculate $\epsilon_0 = \sqrt[3]{2} - 1 \approx 0.26$.

We use (7.5) to calculate the error

$$(8.2) \quad \epsilon_{n+1} = -0.5 \epsilon_n + a_2 \epsilon_n^2 + \dots \approx -0.5 \epsilon_n.$$

The now use (8.2) to make a list of errors with which to check Table 4.1.

Table 8.2

$$\epsilon_1 \approx -0.13$$

$$\epsilon_2 \approx 0.065$$

$$\epsilon_3 \approx -0.0325$$

$$\epsilon_4 \approx 0.01625$$

$$\epsilon_5 \approx -0.008125$$

...

$$\epsilon_{50} \approx 2.31 * 10^{-16}$$

The above Table 8.2 compares nicely to Table 4.1 and shows why 50 iterations are necessary to get 15 decimal digits of the $\sqrt[3]{2}$.

In this case our error relation (8.2) is of the form $|\epsilon_{n+1}| \approx c \epsilon_n$ and thus we can write $|\epsilon_n| = c^n \epsilon_0$. If we want d decimal digits of accuracy then we need

$|\epsilon_n| \leq 0.5 * 10^{-d}$. This leads to the computation

$$c^n \epsilon_0 \leq 0.5 * 10^{-d}$$

$$\log(c^n \epsilon_0) \leq \log(0.5 * 10^{-d})$$

(We are using base ten logarithms.) Using laws of logarithms we solve the above for n and get

$$(8.3) \quad n \geq \frac{d + \log \epsilon_0 - \log 0.5}{-\log c} \approx \frac{d}{-\log c}$$

This simple relation (with $c = 0.5$ and $d = 15$) predicts that we need 50 iterations to obtain 15 decimal digits of accuracy as we saw in Table 8.2.

Case 3: Table 5.1, alpha vs speed of convergence

In Table 5.1 we see the results of using (6.1) to compute $\sqrt[3]{2}$ with various values of alpha. Relation (7.5) gives us the needed error calculation with $A = 2$ and $p = 3$:

$$\epsilon_{n+1} = (\alpha - 2\beta) \epsilon_n - \frac{3\beta}{\sqrt[3]{2}} \epsilon_n^2 + \dots$$

To get quadratic convergence we need the ϵ_n term to vanish. Notice that this happens when $\alpha = 2/3$ and $\beta = 1/3$. The other entries in the table can easily be checked using (8.3) with $c = |\alpha - 2\beta|$ and $d = 15$.

Case 4: Quadratic convergence for any root

We determined by numerical experimentation that the iterations defined by (6.1) would converge quadratically to $\sqrt[p]{A}$ if $\alpha = \frac{p-1}{p}$ and $\beta = \frac{1}{p}$. We can see why this is correct by examining (7.5) and observing that the ϵ_n term vanishes with these choices.

9. References

- [1] H. Anton, *Calculus, Brief Edition*, Wiley, (6 th ed.), 1999.
- [2] D. M. Burton, *History of Mathematics: An Introduction*, McGraw Hill, (3 rd ed.) 1991, p. 79.

- [3] R. J. Knill, *A modified Babylonian algorithm*, Amer. Math. Monthly, 99 (1992), pp. 734-737.