BRICS RS-04-17 M. Goldberg: Computing Logarithms Digit-by-Digit

BRICS

Basic Research in Computer Science

Computing Logarithms Digit-by-Digit

Mayer Goldberg

BRICS Report Series RS-04-17

ISSN 0909-0878 September 2004

BRICS
Department of Computer Science
University of Aarhus
Ny Munkegade, building 540
DK–8000 Aarhus C
Denmark

Telephone: +45 8942 3360
Telefax: +45 8942 3255
Internet: BRICS@brics.dk

BRICS publications are in general accessible through the World Wide Web and anonymous FTP through these URLs:

`http://www.brics.dk`
`ftp://ftp.brics.dk`
This document in subdirectory `RS/04/17/`

Computing Logarithms Digit-by-Digit

Mayer Goldberg[*]

September 2004

Abstract

In this work, we present an algorithm for computing logarithms of positive real numbers, that bares structural resemblance to the elementary school algorithm of long division. Using this algorithm, we can compute successive digits of a logarithm using a 4-operation pocket calculator. The algorithm makes no use of Taylor series or calculus, but rather exploits properties of the radix-d representation of a logarithm in base d. As such, the algorithm is accessible to anyone familiar with the elementary properties of exponents and logarithms.

1 Radixes and Bases

The term *base* has two different meanings, both of which are used in this work, and it is important to distinguish amongst them. The *base of a number system* has to do with the power series representation of a number: If N is written as $a_0 a_1 \cdots a_k$ in base d, then $N = \sum_{j=0}^{k} a_j d^{k-j}$. The *base of a logarithm* has to do with representing a number as the power of another: If $\log_d x = y$, then $d^y = x$. These two distinct meanings of the term *base* are related in this work, so to prevent any ambiguity, we use the term *radix-d* to speak of the base-d representation of a number. Throughout the remainder of this work the term *base* will refer to the base of a logarithm.

2 The Algorithm

Let M be a positive real number. Let the radix-d representation of the logarithm in base d of M be written as $a_0.a_1a_2\cdots$, then:

$$\log_d M = a_0 + \frac{a_1}{d} + \frac{a_2}{d^2} + \frac{a_3}{d^3} \cdots \qquad (1)$$

Therefore

$$\begin{aligned} M &= d^{a_0 + \frac{a_1}{d} + \frac{a_2}{d^2} + \frac{a_3}{d^3} \cdots} \\ &= d^{a_0} \cdot d^{\frac{a_1}{d} + \frac{a_2}{d^2} + \frac{a_3}{d^3} \cdots} \end{aligned}$$

[*]Department of Computer Science, Ben Gurion University, P.O. Box 653, Beer Sheva 84105, Israel.

The central observation is that when we work in the radix-d, the first digit in the base-d expansion, or a_0 in Equation (1), is always readily available. For example, the first digit of $\log_{10} 345$ is 2, because $10^2 \leq 345 < 10^3$. Similarly, the first digit of $\log_{10} 2468$ is 3, because $10^3 \leq 2468 < 10^4$.

Having extracted a_0, we now divide both sides by d^{a_0}, giving

$$\frac{M}{d^{a_0}} = d^{\frac{a_1}{d}+\frac{a_2}{d^2}+\frac{a_3}{d^3}\cdots}$$

Raising both sides to the d-th power, we get

$$\left(\frac{M}{d^{a_0}}\right)^d = d^{a_1+\frac{a_2}{d}+\frac{a_3}{d^2}\cdots} \tag{2}$$

We see from Equation (2) that $a_1.a_2a_3\cdots$ is the radix-d representation of $\log_d\left(\frac{M}{d^{a_0}}\right)^d$, the first digit of which is a_1. Note our previous observation that the first digit of the radix-d representation of the logarithm in base d of some number is readily available, so the process of extracting the logarithm digit-by-digit can proceed.

We presented the algorithm for any choice d of both radix and base of logarithm, in order to show how these two notions are related in the algorithm. For all practical purposes, however, it would seem useful to consider the case where $d = 10$, i.e., computing logarithms in base 10 in decimal notation, or perhaps $d = 2$, where floating-point numbers would be represented in binary notation. The base of the logarithm is not as significant a choice, because it is a simple matter to convert logarithms from one base to another, by multiplying by such constants as $\ln 10$, $\log_2 10$, $\log_2 e$, etc. When computing logarithms on a pocket calculator, it is practical to use rational approximations for such constants. For example, $\ln 10$ can be approximated within $1.58 \cdot 10^{-9}$ by the fraction 5377/12381.[1]

2.1 A worked-out example

In this section, we present a worked-out example of computing $\log_{10} 1234.56$ in decimal (radix-10 notation). The various stages of the algorithm are tabulated below. In computing the first digit of the logarithm of the value in the second column, at each step of the algorithm, we need to count the number of digits to the left of the decimal point, minus one. This figure corresponds to the number of digits in the underlined portion of the value.

[1]A great way to arrive at such rational approximations is using regular continued fractions.[1]

Expression	Value	Next Digit
M	$= 1\underline{234}.56$	$a_0 = 3$
$M_1 = \left(\frac{M}{10^{a_0\ (=3)}}\right)^{10}$	$= 8.2247369382767$	$a_1 = 0$
$M_2 = \left(\frac{M_1}{10^{a_1\ (=0)}}\right)^{10}$	$= 1\underline{416511689}.4063$	$a_2 = 9$
$M_3 = \left(\frac{M_2}{10^{a_2\ (=9)}}\right)^{10}$	$= 3\underline{2}.523825911294$	$a_3 = 1$
$M_4 = \left(\frac{M_3}{10^{a_3\ (=1)}}\right)^{10}$	$= 1\underline{32439}.11735423$	$a_4 = 5$
	$\cdots$	

Hence $\log_{10} 1234.56 = 3.0915\cdots$.

3 Using a pocket calculator

Adapting the algorithm for use on a pocket calculator requires that we address three issues:

- The error involved in the computation
- Raising numbers to the 10-th power on a simple, 4-operation calculator
- Overflow

3.1 Accuracy and error

In theory, had we been able to maintain all the digits obtained from raising numbers to the d-th power, we could have computed logarithms in base d with no loss of accuracy, one digit at a time, to any number of digits. In practice, though, calculators and computers will maintain only so many significant digits, and will round off the rest. As we iterate over the digit-extraction process, the roundoff error will propogate towards the more significant digits. After some iterations, the accumulated error will affect the number of digits to the left of the decimal point, and from that iteration onwards we will be "extracting" incorrect digits. To see how the error builds up, suppose we are computing $\log_d M$ in radix-d, for some integer $d > 1$, and some positive real number M. The error will increase as we raise numbers to the d-th power.

Consider $f(x) = x^d$. For small ϵ, we have

$$\begin{aligned} f(x+\epsilon) &\approx f(x) + \epsilon f'(x) \\ &= x^d + \epsilon d x^{d-1} \end{aligned}$$

Hence for a small error, we have

$$\begin{aligned}(x+\epsilon)^d &\approx x^d + \epsilon d x^{d-1}\\ &= x^d(1+\frac{\epsilon d}{x})\\ &\leq x^d(1+\epsilon d)\end{aligned}$$

The upper bound on the error, the quantity ϵd, propogates the error one digit to the left.

In practice, we can use this algorithm to compute 7-8 correct digits on an 8-digit pocket calculator.

3.2 Raising numbers to the 10-th power

Raising a number to the 10-th power on a simple 4-operation pocket calculator can generally be done without re-entering the number, and without using the memory functions. Most pocket calculators support a feature known as *constant operations*, where given two arguments x, y, and one of the supported binary operations $\otimes \in \{+, -, \times, \div\}$, we can compute the nested, right-associated application

$$\underbrace{x \otimes (x \otimes \cdots (x \otimes}_{n \text{ times}} y) \cdots)$$

The key sequence that computes the above operation on most calculators is given by

$$\boxed{\text{key in } x} \;\otimes\; \otimes\; \boxed{\text{key in } y}\; \underbrace{= \cdots =}_{n \text{ times}}$$

On many calculators, it is even unnecessary to press $\otimes$ twice. When y is not given, the value for x, which appears on the display, is used.

Consequently, we raise a number x to the 10-th power by 9 successive multiplications of x by the *constant* x:

$$\boxed{\text{key in } x} \;\times\; \times\; \underbrace{= \cdots =}_{9 \text{ times}}$$

3.3 Overflow

Computing the 10-th power of a number that is less than 10 requires at least 10 calculator digits to represent, possibly with roundoff errors, but without an overflow error. Most 4-operation pocket calculators carry out calculations up to 8 digits, and hence overflow errors will occur. In this section, we discuss how to resume calculations after such an error.

Most calculators do not clear their display upon overflow errors. Rather, they display the correct digits, and shift the decimal point to the left by as many digits as the display can handle. They would then turn on the error annunciator (usually denoted by a small, capital **E**), and ignore all keyboard input except for the *clear error* key (usually marked $\boxed{\mathsf{CE}}$). Pressing the *clear error* key removes the error condition, and enables further clculations, including constant operations.

For example, $1234567^2 = 1524155677489$. The result has 13 digits (to the left of the decimal point), so the calculation will cause an overflow error on an 8-digit pocket calculator, and result in an error annunciator turned on, and the display showing $\boxed{{\scriptstyle \mathsf{E}}\ 15241.556}$. Note that an 8-digit calculator displays the rounded answer with $13 - 8 = 5$ digits to the left of the decimal point.

In raising to the 10-th power numbers that are less than 10, there are two kinds of overflow situations:

$$\boxed{{\scriptstyle \mathsf{E}}\ d_0.d_1d_2d_3d_4d_5d_6d_7}$$

$$\boxed{{\scriptstyle \mathsf{E}}\ d_0d_1.d_2d_3d_4d_5d_6d_7}$$

representing 9- and 10-digit numbers, the logarithms of which are 8, 9 respectively. When such an overflow condition arises, we note either 8 or 0 in the expansion of the logarithm, reset the decimal point to $\boxed{{\scriptstyle \mathsf{E}}\ d_0.d_1d_2d_3d_4d_5d_6d_7}$, and continue with the algorithm.

4 Related Work

John P. Killingbeck, in his book *The Creative Use of Calculators* [2, Section 4.9], uses probability theory to arrive at an algorithm that amounts to a special case of the algorithm presented herein, where $d = 2$. The output of the Killingbeck's algortihm is a sequence of binary digits that are then converted to decimal and multiplied by $\log 2$ or $\ln 2$ in order to convert the logarithm to a more commonly-used base. The peculiar choice of d in Killingbeck's algortihm may have something to do with the probabilistic argument with which he arrives at his algorithm, and is otherwise unmotivated. The choice of $d = 2$ does, however, seem reasonable for working with representations of floating-point numbers on digital computers.

5 Conclusion

Even though logarithms are taught and used in high school, students are generally unable to compute logarithms, in all but the simplest cases, e.g., when the logarithm is a rational number. Only after they reach college, and study calculus to Taylor series, are they be able to compute the logarithm of any real number in any base. Luckily, this pedagogical wrinkle is easily ironed out, since, as this algorithm shows, logarithms can be computed one digit at a time.

We note that this algorithm is essentially the same as the well-known elementary school algorithm for long-division, where each operation in the long-division algorithm is replaced by a higher operation:

Long Division	Finding a Logarithm
Finding the first digit of the integer quotient	Finding the first digit of the logarithm
Reducing the dividend by the largest integer multiple of the divisor	Dividing the argument by the largest power of d that is smaller than the argument
Multiplying the difference by d, and iterating	Raising the quotient to the power d, and iterating

While the algorithm presented herein is not very efficient, it does offer several pedagogical and computational advantages:

- It assumed no calculus, and rather relies on the most elementary properties of powers and logarithms.
- The algorithm is easy to follow, especially since it is structurally similar to the elementary school algorithm for long division.
- The algorithm lends itself to rapid calculations on a pocket calculator.

Acknowledgements

The author is grateful to BRICS[2] for having hosted him and for providing a stimulating environment. The author is particularly grateful to Saurabh Agarwal, Dani Berend, Olivier Danvy, Ted Eisenberg, Kim Skak Larsen, and Kiril Morozov for their enthusiastic reception of the algorithm, and for their encouragement.

References

[1] Khinchin, Aleksandr Iakovlevich. *Continued Fractions.* Dover Publications, 1997.

[2] John P. Killingbeck. *The Creative Use of Calculators.* Penguin Books, 1981.

[2] Basic Research in Computer Science (`www.brics.dk`), funded by the Danish National Research Foundation.

Recent BRICS Report Series Publications

RS-04-17 Mayer Goldberg. ***Computing Logarithms Digit-by-Digit*****. September 2004. 6 pp.**

RS-04-16 Karl Krukow and Andrew Twigg. ***Distributed Approximation of Fixed-Points in Trust Structures*****. September 2004. 25 pp.**

RS-04-15 Jesús Fernando Almansa. ***Full Abstraction of the UC Framework in the Probabilistic Polynomial-time Calculus ppc*****. August 2004.**

RS-04-14 Jesper Makholm Byskov. ***Maker-Maker and Maker-Breaker Games are PSPACE-Complete*****. August 2004. 5 pp.**

RS-04-13 Jens Groth and Gorm Salomonsen. ***Strong Privacy Protection in Electronic Voting*****. July 2004. 12 pp. Preliminary abstract presented at Tjoa and Wagner, editors,** ***13th International Workshop on Database and Expert Systems Applications*****, DEXA '02 Proceedings, 2002, page 436.**

RS-04-12 Olivier Danvy and Ulrik P. Schultz. ***Lambda-Lifting in Quadratic Time*****. June 2004. 34 pp. To appear in** ***Journal of Functional and Logic Programming*****. This report supersedes the earlier BRICS report RS-03-36 which was an extended version of a paper appearing in Hu and Rodríguez-Artalejo, editors,** ***Sixth International Symposium on Functional and Logic Programming*****, FLOPS '02 Proceedings, LNCS 2441, 2002, pages 134–151.**

RS-04-11 Vladimiro Sassone and Paweł Sobociński. ***Congruences for Contextual Graph-Rewriting*****. June 2004. 29 pp.**

RS-04-10 Daniele Varacca, Hagen Völzer, and Glynn Winskel. ***Probabilistic Event Structures and Domains*****. June 2004. 41 pp. Extended version of an article to appear in Gardner and Yoshida, editors,** ***Concurrency Theory: 15th International Conference*****, CONCUR '04 Proceedings, LNCS, 2004.**

RS-04-9 Ivan B. Damgård, Serge Fehr, and Louis Salvail. ***Zero-Knowledge Proofs and String Commitments Withstanding Quantum Attacks*****. May 2004. 22 pp.**

RS-04-8 Petr Jančar and Jiří Srba. ***Highly Undecidable Questions for Process Algebras*****. April 2004. 25 pp. To appear in Lévy, Mayr and Mitchell, editors,** ***3rd IFIP International Conference on Theoretical Computer Science*****, TCS '04 Proceedings, 2004.**