Semi-Logarithmic Number Systems

Jean-Michel Muller and Arnaud Tisserand

CNRS, Laboratoire LIP
École Normale Supérieure de Lyon
46 Allée d'Italie, 69364 LYON Cedex 07
FRANCE

Alexandre Scherbyna

State Technical University
Cathedral SAPU
31 Povitroflotski prospekt, KIEV, 252037
UKRAINE

Abstract

We present a new class of number systems, called Semi-Logarithmic Number Systems, *that constitute a family of various compromises between floating-point and logarithmic number systems. We propose arithmetic algorithms for the Semi-Logarithmic Number Systems, and we compare these number systems to the classical floating-point or logarithmic number systems.*

1 Introduction

The floating-point number system [5] is widely used for representing real numbers in computers, but many other number systems have been proposed. Among them, one can cite: the logarithmic and sign-logarithm number systems [8, 14, 13, 16, 7, 3, 9] the level-index number system [12, 17, 18], some rational number systems [10], and some modifications of the floating-point number system [19, 11]. Those systems were designed to achieve various goals: e.g. to avoid overflows and underflows, to improve the accuracy, or to accelerate some computations. For instance, the sign-logarithm number system, introduced by Swartzlander and Alexpoulos [14], was designed in order to accelerate the multiplications. As pointed out by the authors, "*it cannot replace conventional arithmetic units in general purpose computers; rather it is intended to enhance the implementation of special-purpose processors for specialized applications*". That number system is interesting for problems where the required precision is relatively low, and where the ratio

$$\frac{\text{number of multiplications}}{\text{number of additions}}$$

is relatively high. Roughly speaking, in such systems, the numbers are represented by their radix-2 logarithms written in fixed-point. The multiplications and divisions are performed by adding or subtracting the logarithms, and the additions and subtractions are performed using tables for the functions $\log_2(1+2^x)$ and $\log_2(1-2^x)$, since:

$$\left\{\begin{array}{l}\log_2(A+B) = \log_2(A) + \log_2(1+2^{\log_2(B)-\log_2(A)}) \\ \log_2(A-B) = \log_2(A) + \log_2(1-2^{\log_2(B)-\log_2(A)})\end{array}\right.$$

The major drawback of the Logarithmic Number System arises when a high level of accuracy is required. If the computations are performed with n-digit numbers, then a straightforward implementation requires a table containing 2^n elements. Interpolation techniques allow the use of smaller tables (see [15, 2, 7]), so that 32-bit logarithmic number systems become feasible. Our purpose in this paper is to present a new number system that allows the use of even smaller tables. That number system will be a sort of compromise between the logarithmic and the floating-point number systems. More exactly, we show a *family* of number systems, parameterized by a number k, and the systems obtained for the two extremal values of k are the floating-point and the logarithmic number systems. With some of those number systems, multiplication and division will be almost as easy to perform as in the logarithmic number system, whereas addition and subtraction will require much smaller tables.

2 The Semi-logarithmic Number Systems

Let k be an integer, let x be a real number different from 0, and define $e_{k,x}$ as the multiple of 2^{-k} satisfying $2^{e_{k,x}} \leq |x| < 2^{e_{k,x}+2^{-k}}$. We immediately find

$$e_{k,x} = \frac{\lfloor 2^k \log_2 |x| \rfloor}{2^k} \tag{1}$$

Define $m_{k,x}$ as:

$$m_{k,x} = \frac{|x|}{2^{e_{k,x}}}$$

from

$$0 \leq \frac{2^k \log_2 |x| - \lfloor 2^k \log_2 |x| \rfloor}{2^k} < \frac{1}{2^k}$$

1063-6889/95 $4.00 © 1995 IEEE

we deduce:

$$1 \leq m_{k,x} = \frac{|x|}{2^{e_{k,x}}} < 2^{\frac{1}{2^k}}$$

Now let us bound the value $2^{\frac{1}{2^k}}$. This value is equal to $e^{\frac{\ln 2}{2^k}}$. If we define α as $1/2^k$, by studying $f(\alpha) = 1 + \alpha - e^{\alpha \ln(2)}$, we easily deduce that $f(\alpha)$ is nonnegative for $0 < \alpha < 1$, that is:

$$2^{\frac{1}{2^k}} \leq 1 + \frac{1}{2^k} \tag{2}$$

for $k \geq 0$. As a consequence, $1 \leq m_{k,x} < 1 + \frac{1}{2^k}$. This leads to the following definitions:

Definition 1 (Canonical Form) *Let k be a positive integer. Every non zero real number x will be represented in the* Canonical form *of the* Semi-Logarithmic Number System (SLNS for short) of Parameter k *by three values s_x, $m_{k,x}$ and $e_{k,x}$ satisfying:*

- $s_x = \pm 1$
- *$e_{k,x}$ is a multiple of* 2^{-k}
- $1 \leq m_{k,x} < 2^{\frac{1}{2^k}}$
- $x = s_x \times m_{k,x} \times 2^{e_{k,x}}$

Definition 2 (General Form) *Let k be a positive integer. Every non zero real number x will be represented in the* General form *of the SLNS of Parameter k by three values s_x, $m_{k,x}$ and $e_{k,x}$ satisfying:*

- $s_x = \pm 1$
- *$e_{k,x}$ is a multiple of* 2^{-k}
- $1 \leq m_{k,x} < 1 + 2^{-k}$
- $x = s_x \times m_{k,x} \times 2^{e_{k,x}}$

The representation of x with n mantissa bits in the *semi-logarithmic number system of parameter k* will be constituted by s_x, $e_{k,x}$ and an n-fractional bit rounding of $m_{k,x}$. In practice, since $1 \leq m_{k,x} < 1 + 2^{-k}$, $m_{k,x}$ has a binary representation of the form:

$$1.\overbrace{\underbrace{0000\ldots000}_{k \text{ zeroes}}\text{xxxx}\ldots\text{xx}}^{n \text{ digits}}$$

Since the first $k+1$ digits of $m_{k,x}$ are known in advance, there is no need to store them (this is similar to the *hidden bit* of some radix-2 floating point systems [5]). Exactly as for normalized floating point representations, a special representation must be chosen for zero. In the following, k is considered implicit, and we write "m_x" and "e_x" instead of "$m_{k,x}$" and "$e_{k,x}$." Some points need to be emphasized:

- If $k = 0$, then the semi-logarithmic system of order k is reduced to a n-mantissa digit floating-point system.
- if $k \geq n$ then the semi-logarithmic system of order k is reduced to a logarithmic number system.
- the canonical form is a *non-redundant* representation. In that form, comparisons are easily performed: if the format of the representation is, from left to right, constituted by the sign, the exponent — which is a multiple of 2^{-k} — and then the mantissa, then comparisons are performed exactly as if the numbers were integers.
- the general form is a *redundant* representation. For instance, if $k = 1$, then $\sqrt{2}$ has two possible representations, namely $1.0000000\ldots \times 2^{0.1}$ — the exponent and mantissa are written in radix 2 — and $1.011010100000100111\ldots \times 2^{0.0}$. Although the comparisons are slightly more difficult with the general form — this is due to the redundancy —, we will prefer that form, because the condition "$1 \leq m_{k,x} < 1 + 2^{-k}$" is easier to check than the condition "$1 \leq m_{k,x} < 2^{\frac{1}{2^k}}$," and because the general form leads to simpler arithmetic algorithms. Anyway, the conversion from the general form to the canonical form is easily performed: assume $s_x \times m_x \times 2^{e_x}$ is in general form. Compare m_x with $\rho_k = 2^{2^{-k}}$. If $m_x < \rho_k$ then the number is already represented in canonical form. If $m_x \geq \rho_k$, then add 2^{-k} to e_x and divide m_x by ρ_k. The obtained result will be the representation of x in canonical form.

So the parameter k makes it possible to choose various compromises between the floating-point number system and the logarithmic number system.

Exactly as in floating-point arithmetic, there are some possible rounding modes. For instance, if we define $\mathcal{Z}(x)$ as the number obtained by rounding m_x (in canonical form) to zero, then we get:

$$\mathcal{Z}(x) = s_x \times \frac{\left\lfloor 2^n \times \frac{|x|}{2^{\lfloor 2^k \log_2 |x| \rfloor / 2^k}} \right\rfloor}{2^n} \times 2^{\lfloor 2^k \log_2 |x| \rfloor / 2^k}$$

Similarly, we define:

- rounding towards $\pm\infty$:

$$\mathcal{I}(x) = s_x \times \frac{\left\lceil 2^n \times \frac{|x|}{2^{\lfloor 2^k \log_2 |x| \rfloor / 2^k}} \right\rceil}{2^n} \times 2^{\lfloor 2^k \log_2 |x| \rfloor / 2^k}$$

- rounding to the nearest:

$$\mathcal{N}(x) = s_x \times \frac{\left\lfloor 2^n \times \frac{|x|}{2^{\lfloor 2^k \log_2 |x| \rfloor / 2^k}} \right\rceil}{2^n} \times 2^{\lfloor 2^k \log_2 |x| \rfloor / 2^k}$$

where $\lfloor u \rceil$ stands for the integer which is the closest to u (a special choice must be taken when $2^n m_{k,x}$ is an odd multiple of 1/2).

3 Basic Arithmetic Algorithms

Now, let us present basic algorithms for multiplication, division, addition, subtraction and comparison. We must notice that as soon as k is larger than $\frac{n}{2}+2$, these algorithms — and especially the multiplication and division algorithms — become very simple.

3.1 Multiplication

Assume we want to multiply $s_x \times m_x \times 2^{e_x}$ by $s_y \times m_y \times 2^{e_y}$ where these values are represented in the Semi-Logarithmic Number System of parameter k (general form). This can be done as follows:

1. Compute $s = s_x \times s_y$, $m = m_x \times m_y$ and $e = e_x + e_y$. s is the sign of the final result, and m has the form $1.000\ldots0m_{k-1}m_km_{k+1}\ldots m_n$. It is worth noting that if $k > n/2$ then the multiplication $m = m_x \times m_y$ can be reduced to an addition ($m_x = 1+\epsilon_1$, and $m_y = 1+\epsilon_2$, with $\epsilon_1, \epsilon_2 < 2^{-n/2}$, therefore $m_x \times m_y = 1 + \epsilon_1 + \epsilon_2 + \epsilon_1\epsilon_2$, and the product $\epsilon_1\epsilon_2$ can be ignored, since it is less than 2^{-n}).

2. The product $m_x \times m_y$ is between 1 and $1+2^{-k+1}+2^{-2k}$, therefore the digits of weight 2^{-k+1} and 2^{-k} of m, say m_{k-1} and m_k, may be different from zero. In such a case, define m^* as the number constituted by the digits of m of weight greater than or equal to 2^{-k-1}. That is to say: $m^* = 1.000\ldots0m_{k-1}m_km_{k+1}$. Look up the values α and 2^α defined below in a small (8-entry) table (with m_{k-1}, m_k and m_{k+1} as address bits):

$$\alpha = \frac{\lfloor -\log_2(m^*) \times 2^k \rceil}{2^k}$$

where $\lfloor u \rceil$ is the integer which is the closest to u.

3. compute $\hat{m} = m \times 2^\alpha$ and $\hat{e} = e - \alpha$. If $\hat{m} \geq 1$, then $\hat{m}$ is the mantissa of the result, while $\hat{e}$ is its exponent. If $\hat{m} < 1$[a] then multiply $\hat{m}$ by $2^{2^{-k}}$ and subtract 2^{-k} from $\hat{e}$: this gives the mantissa and the exponent of the result — by the way, as previously, if $k > n/2 + 2$, these multiplications can be reduced to additions.

[a]Our simulations tend to show that that case is rather unlikely to occur

Proof of the Algorithm

From

$$\alpha = \frac{\lfloor -\log_2(m^*) \times 2^k \rceil}{2^k}$$

we easily deduce

$$-\log_2 m^* - 2^{-k-1} \leq \alpha \leq -\log_2 m^* + 2^{-k-1}$$

therefore

$$\frac{m}{m^*} \times 2^{-2^{-k-1}} \leq m \times 2^\alpha \leq \frac{m}{m^*} \times 2^{+2^{-k-1}}$$

The term $\frac{m}{m^*}$ is equal to $1 + \frac{m-m^*}{m^*}$. $m - m^*$ is less than 2^{-k-1} and we have assumed $m^* \geq 1+2^{-k}$ (if this is not true, then the bits m_{k-1} and m_k are equal to zero, and m is the mantissa of the result). Therefore

$$\frac{m}{m^*} \leq 1 + \frac{2^{-k-1}}{1+2^{-k}}$$

This gives

$$\begin{aligned}
&2^{2^{-k-1}} \times \tfrac{m}{m^*} \leq \left(1+2^{-k-1}\right)\left(1+\tfrac{2^{-k-1}}{1+2^{-k}}\right)\\
&\leq \tfrac{1}{1+2^{-k}}\left(1+2^{-k-1}\right)\left(1+2^{-k}+2^{-k-1}\right)\\
&\leq \tfrac{1}{1+2^{-k}}\left(1+2\times2^{-k}+2^{-2k-1}+2^{-2k-2}\right)\\
&< \tfrac{1}{1+2^{-k}}\left(1+2\times2^{-k}+2^{-2k}\right)\\
&= \tfrac{\left(1+2^{-k}\right)^2}{1+2^{-k}} = 1+2^{-k}
\end{aligned}$$

If $m \times 2^\alpha < 1$, then (since $m/m^* \geq 1$):

$$2^{-2^{-k-1}} \leq m \times 2^\alpha < 1$$

therefore,

$$1 < m \times 2^\alpha \times 2^{2^{-k}} < 2^{2^{-k}} \leq 1 + 2^{-k}$$

3.2 Division

Assume we want to divide $s_x \times m_x \times 2^{e_x}$ by $s_y \times m_y \times 2^{e_y}$, where these values are represented in the Semi-Logarithmic Number System of parameter k (general form). This can be done as follows:

1. Compute $m = \frac{m_x}{m_y}$. If $k > n/2$, this division can be reduced to a subtraction ($m_x = 1 + \epsilon_1$, and $m_y = 1 + \epsilon_2$, with $\epsilon_1, \epsilon_2 < 2^{-n/2}$, therefore $m_x/m_y = 1 + \epsilon_1 - \epsilon_2 - \epsilon_1\epsilon_2 + \epsilon_2^2 + \ldots$, and all the terms but $1 + \epsilon_1 - \epsilon_2$ can be neglected). Even if $k \leq n/2$, the fact that m_y is very close to 1 can be used to accelerate the division process using an iterative division method such as Goldschmidt's algorithm [1]. Also compute $e = e_x - e_y$ and $s = s_x \times s_y$ (s is the sign of the final result).

2. from $1 \leq m_x < 1 + 2^{-k}$ and $1 \leq m_y < 1 + 2^{-k}$, we deduce $\frac{1}{1+2^{-k}} < m = \frac{m_x}{m_y} < 1 + 2^{-k}$, which implies $1 - 2^{-k} < m < 1 + 2^{-k}$. Therefore, m has the form $1.000\ldots00m_{k+1}m_{k+2}\ldots$ or $0.111\ldots1m_{k+1}m_{k+2}\ldots$ If $m_k = 0$, then m is the mantissa of the result, and e remains unchanged. If $m_k = 1$, then

   - Look up the values α and 2^α defined below in a small (2-entry) table (the values only depend on m_{k+1}):

   $$\alpha = \frac{\lfloor -\log_2(m^*) \times 2^k \rceil}{2^k}$$

     where $\lfloor u \rceil$ is the integer which is the closest to u, and $m^* = 0.111\ldots1m_{k+1}$.
   - compute $\hat{m} = m \times 2^\alpha$ and $\hat{e} = e - \alpha$. If $\hat{m} \geq 1$, then $\hat{m}$ is the mantissa of the result, while $\hat{e}$ is its exponent. If $\hat{m} < 1$ then multiply $\hat{m}$ by $2^{2^{-k}}$ and subtract 2^{-k} from the new computed value $\hat{e}$: this gives the mantissa and the exponent of the result — as previously, if $k > n/2 + 2$, these multiplications can be reduced to additions.

3.3 Addition and Subtraction

Assume we want to compute $(s_x \times m_x \times 2^{e_x}) \pm (s_y \times m_y \times 2^{e_y})$, where these values are represented in the Semi-Logarithmic Number System of parameter k (general form). Exactly as in floating-point arithmetic, the basic method consists of "aligning" the mantissas (i.e. rewriting both numbers with the same exponent), adding the aligned mantissas and renormalizing the result. In the following, we assume that e_x is larger than or equal to e_y (if this is not true, exchange both numbers).

1. Define $u = \lfloor e_x - e_y \rceil$ and $v = e_x - e_y - u$ (v satisfies $0 \leq |v| \leq 1/2$, and u is an integer). Perform a u-bit right shift of m_y, then look up the value of $\beta = 2^{-v}$ in a $(k-1)$-address bit table. Multiply the shifted m_y by β. This gives a new value m_y^*.

2. Add (or subtract, depending on the signs) m_x and m_y^*. This gives a value p. From this, deduce the sign of the result. If $p < 0$, replace p by $|p|$. If $p \geq 2$, then perform a 1-bit right shift of p, which gives a value m, and add 1 to e_x, which gives a value e; otherwise, perform a j-bit left shift of p where j is such that after the shift, the most significant "1" of p appears in the position of weight 2^0; this gives a value m, and subtract j from e_x, this gives a value e (if all the digits of p are zero the result of the operation is the special code chosen for the number zero). m has the form $1.m_1\ldots m_k m_{k+1}\ldots m_n$.

3. Define m^* as $1.m_1\ldots m_k m_{k+1}$ and look up the values α and 2^α defined below in a $(k+1)$-address bit table (with m_1, m_2, ..., m_{k+1} as address bits):

$$\alpha = \frac{\lfloor -\log_2(m^*) \times 2^k \rceil}{2^k}$$

4. compute $\hat{m} = m \times 2^\alpha$ and $\hat{e} = e - \alpha$. If $\hat{m} \geq 1$, then $\hat{m}$ is the mantissa of the result, while $\hat{e}$ is its exponent. If $\hat{m} < 1$ then multiply $\hat{m}$ by $2^{2^{-k}}$ and subtract 2^{-k} from the new computed value $\hat{e}$: this gives the mantissa and the exponent of the result — as previously, if $k > n/2 + 2$, this last multiplication can be reduced to an addition.

Provided that $k > n/2+2$, the only "large multiplication" that appear in the arithmetic algorithms is the calculation of $\hat{m} = m \times 2^\alpha$ of the addition/subtraction algorithm (this is a multiplication of two n-bit integers). It is possible to avoid this $n \times n$ multiplication by slightly modifying the algorithm: if, instead of only returning α and 2^α the table used also returns $2^{-\alpha}$, then one can compute $\hat{m}$ as $(m - 2^{-\alpha}) \times 2^\alpha + 1$. It is easy to show that $m - 2^{-\alpha} < 2^{-k+1}$, therefore, the multiplication $(m - 2^{-\alpha}) \times 2^\alpha$ is the multiplication of a $n - k + 1$-bit number by an n-bit number. If $k > n/2$, this leads to a significant reduction in the size of the required multiplier and the time of computation. Moreover, this method does not increase the amount of memory that is required: we only need $n - k + 1$ bits of $2^{-\alpha}$ (since its $k - 1$ most significant bits are zeroed when they are added to m), and we only need the most $n - k + 1$ bits of 2^α, since the influence of its less significant bits is negligible.
The addition/subtraction algorithm is the only algorithm that requires the use of a large table (that contains 2^{k+1} values). This should be compared to the 2^n values that are required when implementing a Logarithmic Number System without interpolations. If a table with 2^{k+1} elements cannot be implemented, one can use two tables with $2^{\frac{k+1}{2}+1}$ elements, and decompose the computation of $\hat{m}$ in two steps:

- Define $j = \frac{k+1}{2}$. In the first step, look up in a $(j+1)$-address bit table (with $m_1, m_2, \ldots, m_{j+1}$ as address bits the values α_1 and 2^{α_1} satisfying:

$$\alpha_1 = \frac{\lceil -\log_2(1.m_1m_2\ldots m_{j+1}) \times 2^k \rceil}{2^k}$$

and compute $m^{(1)} = m \times 2^{\alpha_1}$. One can show that $m^{(1)}$ is between 1 and $1 + 2^{-j+1}$.

- look up in a $(j+1)$-address bit table (with $m_j^{(1)}$, $m_{j+1}^{(1)}$, …, m_{k+1} as address bits) the values α_2 and 2^{α_2} satisfying:

$$\alpha_2 = \frac{\left\lfloor -\log_2\left(1.000\ldots 0 m_j^{(1)} m_{j+1}^{(1)} m_{k+1}^{(1)}\right) \times 2^k \right\rfloor}{2^k}$$

  and compute $\hat{m} = m^{(1)} \times 2^{\alpha_2}$ and $\hat{e} = e - \alpha_1 - \alpha_2$. If $\hat{m} \geq 1$, then $\hat{m}$ is the mantissa of the result, while $\hat{e}$ is its exponent. If $\hat{m} < 1$ then multiply $\hat{m}$ by $2^{2^{-k}}$ and subtract 2^{-k} from the new computed value $\hat{e}$: this gives the mantissa and the exponent of the result.

If tables of size $2^{\frac{k+1}{2}+1}$ are still too large, then both previous steps can be decomposed again.

3.4 comparisons

Assume we want to compare $x = s_x \times m_x \times 2^{e_x}$ and $y = s_y \times m_y \times 2^{e_y}$, where these values are represented in the Semi-Logarithmic Number System of parameter k (general form). We assume that both numbers are positive (if their signs are different, then the comparison is straightforward, and if both numbers are negative, the required modification of the algorithm is obvious). We also assume that $e_x \geq e_y$ (if this is not true, exchange x and y). The comparison can be done as follows:

- If $e_x - e_y > 2^{-k}$ then $x > y$
- if $e_x = e_y$ then $x \geq y$ if and only if $m_x \geq m_y$
- if $e_x - e_y = 2^{-k}$, then multiply m_y by the precomputed value 2^{-2^k} — if $k > n/2$ then this multiplication can be reduced to an addition — this gives a value m_y^*. Then $x \geq y$ if and only if $m_x \geq m_y^*$

4 Static accuracy of the Semi-Logarithmic Number System

In this section we evaluate the Maximum Relative Representation Error (MRRE) and the Average Relative Representation Error (ARRE) [4] of the semi-logarithmic number systems. We perform the computations for the case of the "rounding-to-zero" mode. In the other cases, the computations are very similar. Figure 1 presents the relative error $\frac{x - \mathcal{Z}(x)}{x}$ for x between 1 and 2, $n = 4$, and $k = 2$. For the evaluation of

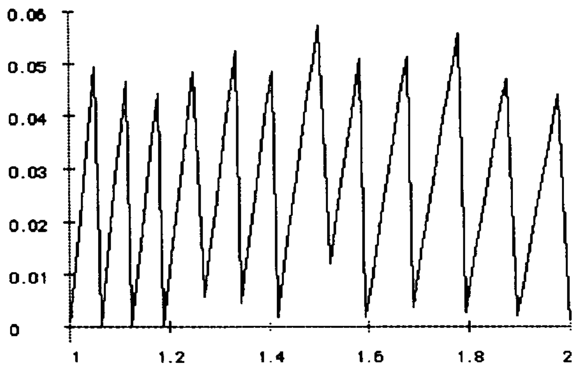

Figure 1: Relative error between 1 and 2 for $n = 4$ and $k = 2$

the average errors, we assume Hamming's *logarithmic distribution* of numbers [6], that is:

$$P(x) = \frac{1}{x \ln 2} \quad \text{where } 1 \leq x < 2$$

4.1 Maximum Relative Representation Error (MRRE)

Assume x is between 1 and 2. We have:

$$\left|\frac{x - \mathcal{Z}(x)}{x}\right| = \frac{x - \frac{\left\lfloor 2^n \times \frac{x}{2^{\lfloor 2^k \log_2 x \rfloor / 2^k}} \right\rfloor}{2^n} \times 2^{\lfloor 2^k \log_2 x \rfloor / 2^k}}{x}$$

This can be rewritten as: $A \times \frac{2^{B/2^k}}{2^n}$, with:

$$\begin{cases} A &= \frac{2^n x}{2^{\lfloor 2^k \log_2 x \rfloor / 2^k}} - \left\lfloor \frac{2^n x}{2^{\lfloor 2^k \log_2 x \rfloor / 2^k}} \right\rfloor \\ B &= \lfloor 2^k \log_2 x \rfloor - 2^k \log_2 x \end{cases}$$

The maximum possible values for A and B are 1 and 0, and it is possible to find x such that A is as close as possible to 1, and B is as close as possible to zero. From this we deduce:

$$MRRE = 2^{-n} \tag{3}$$

As a consequence, the floating-point system, the logarithmic number system, and all the semi-logarithmic number systems lead to the same value of the MRRE.

4.2 Average Relative Representation Error (MRRE)

We want to evaluate

$$ARRE = \int_1^2 \frac{1}{x \ln 2} \times \left|\frac{x - \mathcal{Z}(x)}{x}\right| dx \tag{4}$$

Table 1: ARRE and MRRE of the Semi-logarithmic Number Systems for different values of k

	Rounding to zero		Rounding to nearest	
	MRRE	ARRE	MRRE	ARRE
Floating Point	2^{-n}	0.36×2^{-n}	2^{-n-1}	0.18×2^{-n}
SLNS ($k \geq 2$)	2^{-n}	$2^{-n-1}\ln(2)(1-2^{-k}\ln 2)$	2^{-n-1}	$2^{-n-2}\ln(2)(1-2^{-k}\ln 2)$
SLNS ($k = 4$)	2^{-n}	0.33×2^{-n}	2^{-n-1}	0.17×2^{-n}
Logarithmic	2^{-n}	0.35×2^{-n}	2^{-n-1}	0.17×2^{-n}

Let us define Δ_c as the domain where $\lfloor 2^k \log_2 x \rfloor / 2^k$ equals c. In that domain $\left|\frac{x-\mathcal{Z}(x)}{x}\right|$ is equal to

$$\left(\frac{2^n x}{2^c} - \left\lfloor \frac{2^n x}{2^c} \right\rfloor\right) \times \frac{2^c}{2^n x} \qquad (5)$$

From this, we deduce:

$$\int_{\Delta_c} \frac{1}{x \ln 2} \times \left|\frac{x-\mathcal{Z}(x)}{x}\right| dx \approx \int_{\Delta_c} \frac{1}{x \ln 2} \times \frac{1}{2} \times \frac{2^c}{2^n x} dx$$

Since Δ_c is equal to $\left[2^c, 2^{c+1/2^k}\right)$, we deduce:

$$\int_{\Delta_c} \frac{1}{x \ln 2} \times \left|\frac{x-\mathcal{Z}(x)}{x}\right| dx \approx \frac{2^{c-n-1}}{\ln 2}\left(\frac{1}{2^c} - \frac{1}{2^{c+1/2^k}}\right)$$

The extremal possible values for c are 0 (for $x = 1$) and $\frac{\lfloor 2^k \ln 2 \rfloor}{2^k} \approx \ln 2$ (for $x = 2$).

This gives (by defining i as $c \times 2^k$):

$$ARRE \approx \sum_{i=0}^{\lfloor 2^k \ln 2 \rfloor} \frac{2^{i \times 2^{-k}-n-1}}{\ln 2}\left(\frac{1}{2^{i \times 2^{-k}}} - \frac{1}{2^{(i+1)\times 2^{-k}}}\right)$$

$$= \sum_{i=0}^{\lfloor 2^k \ln 2 \rfloor} \frac{2^{i \times 2^{-k}-n-1}}{\ln 2} \times \frac{2^{2^{-k}}-1}{2^{(i+1)\times 2^{-k}}}$$

Therefore:

$$\begin{aligned} ARRE &\approx \textstyle\sum_{i=0}^{\lfloor 2^k \ln 2 \rfloor} \frac{2^{-2^{-k}-n-1}}{\ln 2} \times \left(2^{2^{-k}}-1\right) \\ &\approx 2^{-n-1} \times \ln(2) \times (1 - 2^{-k}\ln 2) \end{aligned}$$

using $2^{2^{-k}} \approx 1 + 2^{-k}\ln 2$. This approximation is not valid for small values of k (say for $k \leq 1$). For $k = 0$ (which is the case of the floating-point representation), the ARRE is equal to

$$\int_1^2 \frac{1}{x \ln 2} \times \frac{2^{-n}}{2x} dx = \frac{2^{-n-2}}{\ln 2}$$

It must be noted that the ARRE for the semi-logarithmic number system of parameter k is very close to the ARRE of the logarithmic number system (that is to say $2^{-n-1}\ln 2$) as soon as $k \geq 2$.

Table 1 sums up the different values of the maximum and average relative representation error for various cases. An immediate conclusion from this table is that the floating-point, logarithmic and semi-logarithmic number systems lead to approximately the same accuracy.

5 Conclusion

We have proposed a new class of number systems, called ***semi-logarithmic number systems***. They constitute a compromise between the floating point and the logarithmic number systems: if the parameter k is larger than $n/2+2$, multiplication and division are almost as easily performed as in the logarithmic number systems, whereas addition and subtraction require much smaller tables. The best value for k must result from a compromise: if k is large, the tables required for addition may become huge, and if k is small, the algorithms become complicated. Values of k slightly larger than $n/2$ are probably the best choice. With the semi-logarithmic number systems, the average and maximum representation errors are approximately equal (in fact slightly better, but the difference is negligible) to those of the floating-point and the logarithmic number systems. The domain of application of the semi-logarithmic number systems is the same as that of the logarithmic number systems: special purpose processors for solving problems where the ratio

$$\frac{\text{number of multiplications}}{\text{number of additions}}$$

is relatively high.

References

[1] S.F. Anderson, J.G. Earle, R.E. Goldschmidt, and D.M. Powers. The IBM 360/370 model 91: floating-point execution unit. *IBM Journ. of Res. and Dev.*, January 1967. Reprinted in

E.E. Swartzlander, Computer Arithmetic, Vol. 1, IEEE Computer Society Press Tutorial, 1990.

[2] M.G. Arnold, T.A. Bailey, J.R. Cowles, and J.J. Cupal. Redundant Logarithmic Number Systems. In M.D. ercegovac and E.E. Swartzlander, editors, *9th Symposium on Computer Arithmetic*, pages 144–151, Santa Monica, CA, Sept. 1989. IEEE Computer Society Press.

[3] M.G. Arnold, T.A. Bailey, J.R. Cowles, and M.D. Winkel. Applying features of IEEE 754 to sign/logarithm arithmetic. *IEEE Transactions on Computers*, 41(8):1040–1050, August 1992.

[4] W.J. Cody. Static and dynamic numerical characteristics of floating-point arithmetic. *IEEE Transactions on Computers*, C-22(6):598–601, June 1973.

[5] D. Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys*, 23(1):5–47, March 1991.

[6] R.W. Hamming. On the distribution of numbers. *Bell Systems Technical Journal*, 49:1609–1625, 1970. Reprinted in E.E. Swartzlander, Computer Arithmetic, Vol. 1, IEEE Computer Society Press Tutorial, 1990.

[7] H. Henkel. Improved addition for the logarithmic number systems. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 37:301–303, 1989.

[8] N.G. Kingsbury and P.J.W. Rayner. Digital filtering using logarithmic arithmetic. *Electronic Letters*, 7:56–58, 1971. Reprinted in E.E. Swartzlander, Computer Arithmetic, Vol. 1, IEEE Computer Society Press Tutorial, 1990.

[9] D.M. Lewis. An accurate LNS arithmetic using interleaved memory function interpolator. In M.J. Irwin E.E. Swartzlander and G. Jullien, editors, *11th Symposium on Computer Arithmetic*, pages 2–9, Los Alamitos, CA, June 1993. IEEE Computer Society Press.

[10] D.W. Matula and P. Kornerup. Finite Precision Rational Arithmetic: Slash Number Systems. *IEEE Transactions on Computers*, C-34(1):3–18, January 1985.

[11] S. Matsui and M. Iri. An overflow/underflow free floating-point representation of numbers. *Journal of Information Processing*, 4(3):123–133, 1981. Reprinted in E.E. Swartzlander, Computer Arithmetic, Vol. 2, IEEE Computer Society Press Tutorial, 1990.

[12] F.W.J. Olver. A closed computer arithmetic. In *8th IEEE Symposium on Computer Arithmetic.* IEEE Computer Society Press, May 1987. Reprinted in E.E. Swartzlander, Computer Arithmetic, Vol. 2, IEEE Computer Society Press Tutorial, 1990.

[13] T. Stouraitis and F.J. Taylor. Floating-point to logarithmic encoder error analysis. *IEEE Transactions on Computers*, C-37:858–863, 1988.

[14] E.E. Swartzlander and A.G. Alexpoulos. The sign-logarithm number system. *IEEE Transactions on Computers*, December 1975. Reprinted in E.E. Swartzlander, Computer Arithmetic, Vol. 1, IEEE Computer Society Press Tutorial, 1990.

[15] F.J. Taylor. An Extended Precision Logarithmic number System. *IEEE Transactions on Acoustics, Speech, Signal Proc.*, 31:231, 1983.

[16] F.J. Taylor, R. Gill, J. Joseph, and J. Radke. A 20 bit logarithmic number system processor. *IEEE Transactions on Computers*, 37(2):190–200, February 1988.

[17] P.R. Turner. Implementation and analysis of extended SLI operations. In P. Kornerup and D. Matula, editors, *proceedings of the 10th IEEE Symposium on Computer Arithmetic*, pages 118–126. IEEE Computer Society Press, June 1991.

[18] P.R. Turner. Complex SLI arithmetic: representation, algorithms, and analysis. In M.J. Irwin E.E. Swartzlander and G. Jullien, editors, *11th Symposium on Computer Arithmetic*, pages 18–25, Los Alamitos, CA, June 1993. IEEE Computer Society Press.

[19] H. Yokoo. Overflow/underflow-free floating-point number representations with self-delimiting variable-length exponent fields. *IEEE Transactions on Computers*, 41(8):1033–1039, August 1992.