MATLAB Simulator of Level-Index Arithmetic

Mantas Mikaitis

School of Computing, University of Leeds, Leeds, United Kingdom

Abstract—**Level-index arithmetic appeared in the 1980s. One of its principal purposes is to abolish the issues caused by underflows and overflows in floating point. However, level-index arithmetic does not expand the set of numbers but spaces out the numbers of large magnitude even more than floating-point representations to move the infinities further away from zero: gaps between numbers on both ends of the range become very large. We revisit level index by presenting a custom precision simulator in MATLAB. This toolbox is useful for exploring performance of level-index arithmetic in research projects, such as using 8-bit and 16-bit representations in machine learning algorithms where narrow bit-width is desired but overflow/underflow of floating-point representations causes difficulties.**

Index Terms—**level-index arithmetic, floating-point arithmetic, number systems**

I. INTRODUCTION

In a level-index (LI) arithmetic of Clenshaw and Olver [1] a positive number $x \in \mathbb{R}$ is represented with $l \in \mathbb{N}$ (a level) and $f \in [0, 1)$ (an index) as

$$x = e^{e^{\cdot^{\cdot^{\cdot^{e^f}}}}}. \tag{1}$$

Here

$$f = \ln(\ln(\cdots \ln(x) \cdots)). \tag{2}$$

The exponentiation or the logarithm are taken l times. Numbers $x < 1$ could be represented by taking $l = 0$ and $f = x$, but more of them can be represented by the symmetric level-index (SLI) system by Clenshaw and Turner [3] which adds a reciprocal sign to (1) used for $x < 1$. The term "symmetric" presumably refers to the numbers of values represented in the ranges $x \in (0, 1)$ and $x \in (1, +\infty)$ being the same; the ordinary LI arithmetic that does not use the reciprocal sign is also symmetric, but with respect to zero when taking into account the negative axis.

Formally, a nonzero real number x in the SLI systems is represented by a number $\zeta = l + f$ and the following relations [3]:

$$x = s(x)\phi(\zeta)^{r(x)}, \tag{3}$$

where $s(x) = \pm 1$ is the sign of x, $r(x) = \pm 1$ is the reciprocal sign defined by

$$r(x) = \begin{cases} +1, & \text{if } |x| \geq 1, \\ -1, & \text{if } |x| < 1, \end{cases} \tag{4}$$

and

$$\phi(\zeta) = \begin{cases} \zeta, & \text{if } 0 \leq \zeta < 1, \\ e^{\phi(\zeta-1)}, & \text{if } \zeta \geq 1. \end{cases} \tag{5}$$

Here (5) computes (1) given a LI number. Note that (5) produces $\lfloor \zeta \rfloor = l$ exponentials, with the final exponent $\zeta - \lfloor \zeta \rfloor = f$ as required by the definition of the LI systems.

To construct ζ Clenshaw and Olver [1] propose

$$\Psi(x) = \begin{cases} x, & \text{if } 0 \leq x < 1, \\ 1 + \Psi(\ln(x)), & \text{if } x \geq 1, \end{cases} \tag{6}$$

which is similar to (2) except that the level is also included with 1 being added on every recursive step.

Note that precision p does not come in anywhere in this definition, unlike the floating-point representation that usually contains p. Of course, p plays a role in implementing the quantisation of the index f.

A. Previous results

Turner [13] demonstrates a Pascal software package that simulates a SLI format with 3 level bits and 27 index bits.

Lozier and Olver [10], [8] show that LI system is *closed* which means that, unlike in floating point, it is impossible to produce numbers that lie outside the representable range with the basic operations, except division by zero. The authors [10, Sec. 3] also explain that levels beyond 6 bits will not be entered in practice by addition, subtraction, multiplication and division, and therefore that 3 bits are enough for the level. Furthermore, Olver [10, Sec. 4] writes that LI systems are free from "wobbling precision", a feature of floating point whereby a real number x rounded to a floating-point system with precision p is $\text{fl}(x) = x(1 + \delta)$ and the error δ can be anywhere between -2^{-p} and 2^{-p}. Olver also mentions that LI is more precise than floating-point for $x < 2^{11}$ in 32 bits and for $x < 2^{44}$ in 64 bits, but less precise beyond $x > 2^{18}$ and $x > 2^{70}$ for 32- and 64-bit representations, respectively.

Demmel [4] argues that LI and other similar arithmetics such as the one by Iri and Matsui [9] that aim to remove the possibility of overflow, overall do not result in improvements since more care is needed when computing with very big highly inaccurate quantities. This is in contrast with floating point that returns infinities allowing to detect overflows.

Shen and Turner [12] explore a hybrid floating point and LI arithmetic. They propose to do most computations in standard binary64 [6] arithmetic, but switch to LI once certain bounds are reached on the input arguments to the four basic arithmetic operations.

Kwak and Swartzlander [7] propose a hardware implementation of LI arithmetic and demonstrate area reduction with a minor increase in the timing of the circuit, compared with the previous approach by Olver and Turner [11].
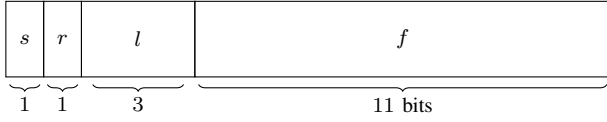
Fig. 1. Layout of a possible 16-bit SLI encoding: sli-3.11.

II. THE ENCODING OF LEVEL-INDEX NUMBERS

We need to encode the LI numbers of (3) in a limited precision word length for use on the digital computers. A sign $s(x)$ can be one bit, the reciprocal sign $r(x)$ can also be one bit, the level l, the authors recommend, does not have to be more than 3 bits to "virtually abolish overflow from everyday work" [1], and the index f can be as precise as possible and represented in fixed point. We will refer to a sli encoding with a k-bit level and a p-bit index as sli-$k.p$.

Figure 1 shows an encoding of a 16-bit binary SLI representation with a 3-bit level and an 11-bit index. We have placed the reciprocal sign $r(x)$ to the left of the level. This way the sequence of representable numbers starts from the smallest number, representing $r(x) = -1$ by setting the reciprocal bit to zero. Then, when the encoding bit pattern is incremented by 1, it transitions through the levels and eventually $r(x) = 1$ is set when the representation for number 1 is reached.

III. SMALL LEVEL-INDEX SYSTEMS

Here we compare an unsigned 5-bit SLI representation with an unsigned 5-bit binary floating-point ("toy") system used for demonstration by Higham [5]. In the interest of saving space we don't include the negative axes—the representations are symmetrical with respect to zero. In floating point, numbers are represented with $\pm\beta^{e-p+1} \times m$. Here β is a base, p is precision, and $e_{min} \le e \le e_{max}$ is the exponent. The exponent is usually encoded with a bias: $E = e + e_{max}$. In IEEE 754 [6] $e_{min} = 1 - e_{max}$. The significand m satisfies $0 \le m \le \beta^p - 1$, but the normalized nonzero numbers are assumed to have $m \ge \beta^{p-1}$ whilst the subnormal values have $m \le \beta^{p-1} - 1$ and a fixed exponent $e = e_{min}$. Stored significand M omits the most significant bit of m. We will assume the IEEE 754 floating-point encoding, including representing infinities, subnormals and not-a-number (NaNs).

For the 5-bit SLI representation, we consider 1 and 2 bits for the level. Since the encoding for level zero is not required, we use the level encoding 00 for representing level 1 and therefore have levels 1 to 4. Similarly for 1-bit level SLI representations: 0 encodes level 1 and 1 encodes level 2. The authors of LI mention that it could instead be used for representing special values [3], perhaps values equivalent to NaNs or infinities in floating point. For the reciprocal bit sign, we use 0 for $r(x) = -1$ and 1 for $r(x) = 1$ and store it on the left of the level bits in order to have small values represented by the lower half of the set of representable binary patterns.

Figure 2 shows the layouts of 5-bit floating-point representation and two different 5-bit SLI representations. Table I lists the 32 possible values representable by the three systems. The following observations can be made from this table.
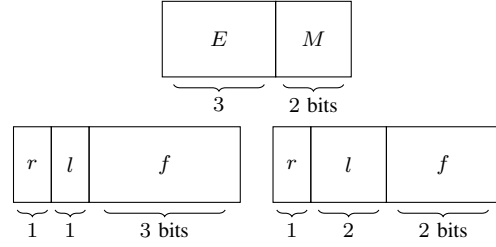

Fig. 2. Layout of an unsigned 5-bit toy floating-point system [5] (top) and two unsigned SLI systems: sli-1.3 (bottom left) and sli-2.2 (bottom right).

- SLI systems have two representations for 1 and no representation for 0. One of the bit patterns for 1 could be used for representing zero. We have used all zeros to represent the zero.
- SLI systems represent decreasing numbers with the increasing binary patterns from 00000 to 01111, which happens because of the reciprocal rule for representing small values below 1. This could be changed by inverting the index bits on conversion to and from, if $r(x) = 0$. Then 00000 could be used to represent the zero, with the following number 00001, when inverted, representing the smallest representable nonzero number. We did not implement this in v0.1 of the toolbox to keep the encoding closer to the definition (3).
- The SLI system with the 1-bit level does not offer a wider dynamic range than the 5-bit floating-point system.
- The SLI system with the 2-bit level offers a wider dynamic range than the binary64 [6] representation.

IV. ARITHMETIC WITH LEVEL-INDEX NUMBERS

The algorithms for LI arithmetic are shown by Clenshaw and Olver [2] while the modifications required for the SLI systems are investigated by Clenshaw and Turner [3]. We provide key highlights to demonstrate what is involved in implementing LI arithmetic; readers should refer to Clenshaw, Olver, and Turner for complete algorithms.

Take $X = l + f$, $Y = m + g$, $Z = n + h$, $X \ge Y \ge 0$, the LI numbers with corresponding levels l, m and n, indices f, g and h, and $\phi(X) \pm \phi(Y) = \phi(Z)$. In the standard LI arithmetic, addition and subtraction operations require three sequences $a_j = 1/\phi(X - j)$, $b_j = \phi(Y - j)/\phi(X - j)$, and $c_j = \phi(Z - j)/\phi(X - j)$. Sequences terminate as soon as $c_j < a_j$, and additional calculations on c_j provide the level and index values of the final result [2] (rounding or chopping to required precision). These sequences are short because for a_j and c_j j goes up to the level of X while for b_j up to the level of Y. Clenshaw and Olver [2] provide

$$a_{l-1} = e^{-f}, \ a_{j-1} = e^{-1/a_j},$$

$$b_{m-1} = a_{m-1}e^g, \ b_{j-1} = e^{-(1-b_j)/a_j} \text{ (if } m \ge 1\text{), and}$$

$$c_0 = 1 - b_0, \ c_j = 1 + a_j \ln(c_{j-1}).$$

If $m = 0$, we compute $b_0 = a_0 g$ instead of the expression above. For addition $c_0 = 1 + b_0$. The sequence c_j is stopped

	FP	sli-1.3	sli-2.2
00000	0	$(e^0)^{-1} = 1$	$(e^0)^{-1} = 1$
00001	0.0625	$(e^{0.125})^{-1} \approx 0.8825$	$(e^{0.25})^{-1} \approx 0.7788$
00010	0.125	$(e^{0.25})^{-1} \approx 0.7788$	$(e^{0.5})^{-1} \approx 0.6065$
00011	0.1875	~ 0.6873	~ 0.4724
00100	0.25	~ 0.6065	$(e^{e^0})^{-1} \approx 0.3679$
00101	0.3125	~ 0.5353	~ 0.2769
00110	0.375	~ 0.4724	~ 0.1923
00111	0.4375	~ 0.4169	~ 0.1204
01000	0.5	$(e^{e^0})^{-1} \approx 0.3679$	$(e^{e^{e^0}})^{-1} \approx 0.06599$
01001	0.625	$(e^{e^{0.125}})^{-1} \approx 0.322$	~ 0.02702
01010	0.75	$(e^{e^{0.25}})^{-1} \approx 0.2769$	~ 0.0055
01011	0.875	~ 0.2334	$\sim 2.4 \times 10^{-4}$
01100	1	~ 0.1923	$(e^{e^{e^{e^0}}})^{-1} \approx 2.6 \times 10^{-7}$
01101	1.25	~ 0.1544	$\sim 8.4 \times 10^{-17}$
01110	1.5	~ 0.1204	$\sim 1.7 \times 10^{-79}$
01111	1.75	~ 0.0908	$\sim 10^{-1758}$
10000	2	$(e^0)^1 = 1$	$(e^0)^1 = 1$
10001	2.5	$(e^{0.125})^1 \approx 1.1331$	$(e^{0.25})^1 \approx 1.284$
10010	3	$(e^{0.25})^1 \approx 1.284$	$(e^{0.5})^1 \approx 1.6487$
10011	3.5	~ 1.455	~ 2.117
10100	4	~ 1.6487	$(e^{e^0})^1 \approx 2.7183$
10101	5	~ 1.8682	~ 3.6111
10110	6	~ 2.117	~ 5.2003
10111	7	~ 2.3989	~ 8.3062
11000	8	$(e^{e^0})^1 \approx 2.7183$	$(e^{e^{e^0}})^1 \approx 15.1533$
11001	10	$(e^{e^{0.125}})^1 \approx 3.1054$	~ 37.0085
11010	12	$(e^{e^{0.25}})^1 \approx 3.6111$	~ 181.3313
11011	14	~ 4.2844	~ 4048.8237
11100	$+\infty$	~ 5.2	$(e^{e^{e^{e^0}}})^1 \approx 3.8 \times 10^6$
11101	NaN	~ 6.4769	$\sim 1.18 \times 10^{16}$
11110	NaN	~ 8.306	$\sim 5.6387 \times 10^{78}$
11111	NaN	~ 11.0108	$\sim 10^{1758}$

when $c_j < a_j$, at which point $n = j$ and $h = c_j/a_j$. If $c_j \geq a_j$ for $j = 0, \ldots, l-1$, then $n = l$ and $h = f + \ln(c_{l-1})$ [2].

Multiplication and division are straightforward [2]: with extra manipulations of arguments we turn the operations into addition or subtraction and therefore reuse the sequences above. For example, if $m > 0$ then $n > 0$ and $\phi(X)\phi(Y) = \phi(Z) = e^{\phi(X-1)}e^{\phi(Y-1)} = e^{\phi(Z-1)}$ allows to write $\phi(X-1) + \phi(Y-1) = \phi(Z-1)$. We can then do the addition and increase the level of $Z - 1$ by one. Further details are in [2].

Arithmetic for SLI systems requires a few modifications since there is no level zero and the reciprocal sign has to be taken into account. These modifications are described by Clenshaw and Turner [3].

V. MATLAB SYMMETRIC LEVEL INDEX SLI.M

We have implemented a simulator for the SLI arithmetic [3] in MATLAB. Version 0.1 is available on GitHub[1]. The file sli.m defines a sli object, with the following properties.

- level_bits: number of bits assigned to the level (p_l). By default it is set to 2.

[1]https://github.com/north-numerical-computing/level-index-simulator.git

- index_bits: number of bits assigned to the index (p_i). By default it is set to 12.
- sign: sign bit. 0 for $s(x) = 1$ and 1 for $s(x) = -1$.
- reciprocal: 1 for $r(x) = 1$ and 0 for $r(x) = -1$.
- level: level, stored as binary64, limited to $[1, 2^{p_l}]$. Since it is a positive integer it could be stored as a 64-bit integer.
- index: index, stored as binary64, rounded to a fixed-point representation with machine epsilon $\varepsilon = 2^{-p_i}$ using MATLAB's round() (round-to-nearest ties-to-away, a default rounding mode). This value could be stored as a 64-bit integer in fixed-point representation.
- value: a binary64 image of the stored LI number, constructed using (5).

Below is an example use of sli in MATLAB.

```
>> x=sli
[...]
>> x=x.set_val(pi)
x =
  sli with properties:
    level_bits: 2
    index_bits: 12
          sign: 0
    reciprocal: 1
         level: 2
         index: 0.135253906250000
         value: 3.141899100868418
>> x*x
ans =
  sli with properties:
    level_bits: 2
    index_bits: 12
          sign: 0
    reciprocal: 1
         level: 2
         index: 0.828369140625000
         value: 9.870807937639510
```

There are two ways to define a sli object: by specifying a binary64 quantity or by explicitly specifying the level and index values. The first method uses (6) to convert a binary64 value to a LI value, with rounding to nearest for fitting the index into the specified number of bits. See the example code within the repository for more detail.

VI. EXPERIMENTS

Figures 3 and 4 show the accuracy of a 16-bit SLI arithmetic sli-2.12 compared with the binary16 and bfloat16 floating-point representations, respectively. The accuracy was measured by comparing with binary64 in a narrow range of numbers around zero with a step size between the adjacent input samples of 10^{-5}, computing the relative error. The step size was chosen so that it is small enough to capture many values but big enough to visualize the errors in a plot.

Figure 5 and 6 show the relative backward error for matrix-vector multiplication Ax with A drawn from the two distributions shown and $x \in (0,1)^n$ for $n = [10, 10^4]$. Binary16 demonstrates higher accuracy, but it overflows when $A \in (0, 100)^{n \times n}$ whilst sli-2.12 continues computing. On the
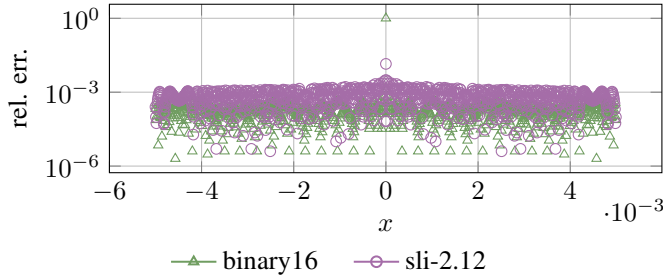
Fig. 3. Relative accuracy of binary16 and a 16-bit level-index representation compared with binary64.
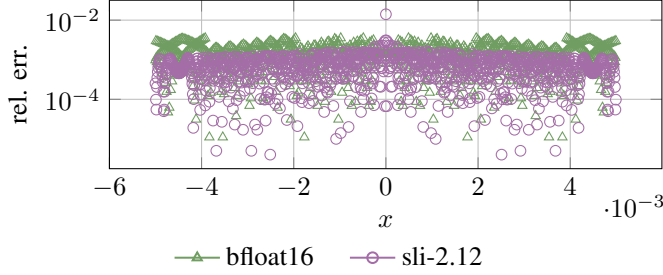

Fig. 4. Relative accuracy of bfloat16 and a 16-bit level-index representation compared with binary64.

other hand bfloat16 does not overflow in this particular experiment; sli-2.12 has better or equivalent accuracy compared with bfloat16.
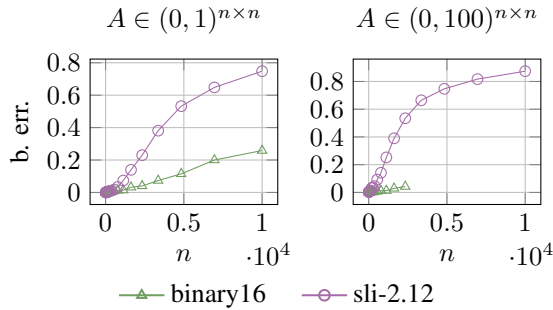

Fig. 5. Backward error in Ax with binary16 and sli compared with binary64.

VII. CONCLUSION

SLI arithmetic simulator is presented which enables the community to experimentally study this number system. In v0.1 we implemented most of the operators[2] for the `sli` objects. Operators `mrdivide`, `mldivide`, `power`, `mpower`, `and`, `or`, and `not` are not yet implemented; the toolbox does not at present fully work in Octave. We plan extensions in the future versions of the toolbox. Our goal is for the toolbox to act as an easy method for testing the accuracy of modern

[2]https://uk.mathworks.com/help/matlab/matlab_oop/implementing-operators-for-your-class.html
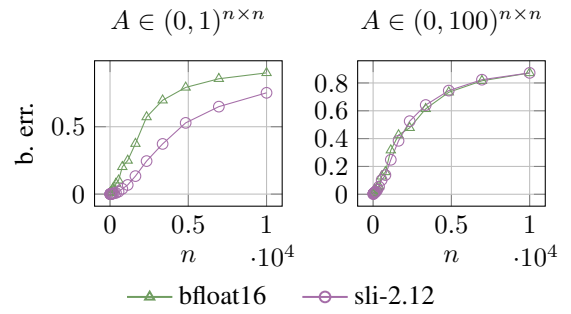

Fig. 6. Backward error in Ax with bfloat16 and sli compared with binary64.

algorithms in SLI arithmetic which in turn may drive hardware architects to have another look at its implementation.

VIII. ACKNOWLEDGEMENTS

REFERENCES

[1] C. W. CLENSHAW AND F. W. J. OLVER, *Beyond floating point*, Journal of the ACM, 31 (1984), p. 319–328.

[2] ———, *Level-index arithmetic operations*, SIAM Journal on Numerical Analysis, 24 (1987), pp. 470–485.

[3] C. W. CLENSHAW AND P. R. TURNER, *The symmetric level-index system*, IMA Journal of Numerical Analysis, 8 (1988), pp. 517–526.

[4] J. W. DEMMEL, *On error analysis in arithmetic with varying relative precision*, in 1987 IEEE 8th Symposium on Computer Arithmetic (ARITH), Como, Italy, May 1987, pp. 148–152.

[5] N. J. HIGHAM, *The mathematics of floating-point arithmetic*, LMS Newsletter, 493 (2021), pp. 35–41.

[6] *IEEE Standard for Floating-Point Arithmetic, IEEE Std 754-2019 (revision of IEEE Std 754-2008)*, July 2019.

[7] J.-H. KWAK AND E. SWARTZLANDER, *An implementation of level-index arithmetic based on the low latency CORDIC system*, in 32nd Asilomar Conference on Signals, Systems and Computers, 1998, pp. 208–212.

[8] D. W. LOZIER AND F. W. J. OLVER, *Closure and precision in level-index arithmetic*, SIAM Journal on Numerical Analysis, 27 (1990), pp. 1295–1304.

[9] S. MATSUI AND M. IRI, *An overflow/underflow-free floating-point representation of numbers*, Journal of Information Processing, 4 (1981), pp. 123–133.

[10] F. W. J. OLVER, *A closed computer arithmetic*, in 1987 IEEE 8th Symposium on Computer Arithmetic (ARITH), Como, Italy, May 1987, pp. 139–143.

[11] F. W. J. OLVER AND P. R. TURNER, *Implementation of level-index arithmetic using partial table look-up*, in 1987 IEEE 8th Symposium on Computer Arithmetic (ARITH), Como, Italy, 1987, pp. 144–147.

[12] X. SHEN AND P. TURNER, *A hybrid number representation scheme based on symmetric level-index arithmetic*, in Proceedings of the 2006 International Conference on Scientific Computing, CSC 2006, Las Vegas, Nevada, USA, June 26-29, 2006, H. R. Arabnia, ed., CSREA Press, 2006, pp. 118–123.

[13] P. TURNER, *A software implementation of SLI arithmetic*, in Proceedings of 9th Symposium on Computer Arithmetic, Santa Monica, CA, USA, 1989, pp. 18–24.