Encoding floating point numbers to shorter integers

Kiyoshi Yoneda
Fukuoka University
`yoneda@fukuoka-u.ac.jp`

Charles Childers
ForthWorks
`crc@forthworks.com`

Draft May 23, 2018

Abstract

A scheme for encoding floating point numbers to shorter integers is proposed and evaluated. The scheme retains a high accuracy near zero while retaining a low relative error away from zero. To encode 64-bit floats to 32-bit integers, the dynamic range can be taken to be from 10^{-9} to 10^{9}, with a 32-bit accuracy near zero and a 16-bit accuracy near 1. The dynamic range and the accuracy may be adjusted by a trade-off parameter. The motivation for devising this scheme comes from deployment of minimal virtual machines for embedded systems programming.

MSC codes

65 Numerical Analysis, 65Y04 Algorithms for computer arithmetic, etc.

1 Introduction

In numerical computation numbers are normally stored in a floating point (FP) format ready to be processed by a floating point unit (FPU). However, the practice is not always economical in resource-constrained computation. This report proposes a way to encode FP numbers, occupying typically 64 bits or more, to shorter integers of typically 32 bits or less. The motivation comes from virtual machine (VM) simplification.

The proposal is to encode the FP number to its square root. Consider a floating point (FP) number with a single decimal digit mantissa, say `1e-2` = 0.01, to be stored in a memory space that permits only one fractional digit. If fixed point representation is employed, by rounding 0.01 to a single fractional digit, its approximate is stored as 0.0 . On the other hand, by encoding the FP number to its square root, 0.01 may be stored as $0.01^{\frac{1}{2}} = 0.1$. Converting the stored numbers back to FP, the fixed point version is decoded to `0e-2` $= 0.00$, while the square root version is decoded more precisely to 0.1^2 = `1e-2` $= 0.01$. This report exploits the above property for FP encoding and decoding conversion (FPC) aiming to fit FP numbers to integers of shorter or equal length.

Embedded programming delegates decisions to resource-constrained devices. Consider for instance vending machines. Each machine has a microcomputer to control basic tasks of receiving the payment in a variety of forms and releasing the chosen merchandise. If a machine were to sell perishable goods it ought to be able to change the selling price over time taking into account the validity of goods in stock and expected demand until next inventory update. Autonomy is better than remote control to avoid communication bottleneck and for quick response based on information local to the machine. To make decisions rationally rather than by rule-based reflex, numerical computation is indispensable, for which FP is convenient.

The problem of FPC arises in deployment of minimal VMs designed to operate over short memory cells. It is a standard strategy to use VMs to secure software portability spanning various hardware alternatives. This is important for systems built with inexpensive microcontrollers which come in a large variety of architectures.

An application to be built on a VM involves numerical calculation requiring a dynamic range wider than 32-bit fixed point numbers can handle. An obvious solution to keep the VM simple is to build it to operate uniformly over 64-bit cells to accommodate all basic data types, integers and FP numbers alike. This keeps software simple but is expensive in terms of memory space.

The VM adopted, called Nga (Childers, 2018), is in the style of a minimal instruction set computer, which is an indication of high portability. The VM comes without input/output (I/O) instructions; a programming language called Retro built on top of Nga interfaces between the VM and applications providing I/O and other functionalities including FP. The Retro processor, which has an Nga image built in, is deemed small enough for resource-constrained computation.

The numbers Nga operates upon are almost exclusively signed integers: all FP numbers live confined on a small FP stack in Retro; they cannot be stored in regular integer-size cells. To handle FP directly using Nga a new layer of software such as FP variables and arrays would have to be built as a part of Retro. That would increase the software complexity as well as its size, not to mention the work involved.

Fitting FP numbers to the integer-size cells seemed like a better solution because by adding a FPC, Retro's integrity with Nga is intact while numerical calculation can be carried out in a FPU external to the VM, provided that the inevitable loss in accuracy, taking place whenever a float is encoded to integer, remains tolerable. Figure 1 is a conceptual diagram.

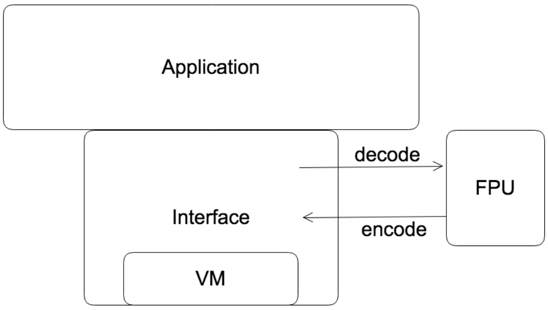

Figure 1: VM and FPU

In this report we assume that FP numbers are encoded into shorter integer-size cells:

FP : consists typically of 64 bits, with 56-bit mantissa and 8-bit exponent, but may be bigger or smaller.
Cell : typically 32 bits, but often smaller.
FPU : available either as hardware or as software library including conversion of integer to FP and back.

Desiderata for a FPC are:

Individual : encoding and decoding should depend only on the number being converted without referring to other numbers in the data set.
Fast : involves no slow functions.
High accuracy near zero : retains a high accuracy around the origin.

The FPC proposed herein exploits the last entry of the list which permits lower accuracy away from the origin. This is reasonable considering that most numbers appear near zero rather than towards the end of the dynamic range.

The remainder of this report is organized as follows. Section 2 considers conditions FPC should satisfy, including a formalization of “high accuracy near zero.” Section 3 proposes a new FPC scheme satisfying the conditions and examines its properties. Section 4 describes an implementation. Section 5 concludes with an observation regarding implementation of the method of least squares.

2 Decoding functions

To consider design alternatives for FPC it is convenient to model the FP numbers as the reals $\mathbb{R}$. The set of numbers encoded to integer-size cells is modeled as

$$\mathbb{U} = \{-M,\ -(M-1),\ \cdots,\ -1,\ 0,\ 1,\ \cdots,\ M-1,\ M\}$$

where M is the largest integer a cell can hold. The intention is that $\mathbb{U}$ approximates $\mathbb{R}$: while both $\mathbb{U}$ and $\mathbb{R}$ have a common notion of '$\leq$', $\mathbb{U}$'s understanding is coarser than $\mathbb{R}$'s. Under this model the problem is to find a pair of functors (e.g. (Leinster, 2016)) $\mathrm{R} : \langle \mathbb{U}, \leq \rangle \to \langle \mathbb{R}, \leq \rangle$ and $\mathrm{U} : \langle \mathbb{U}, \leq \rangle \leftarrow \langle \mathbb{R}, \leq \rangle$ such that

$$\text{in} \qquad \begin{tikzcd} \mathbb{U} \arrow[r, shift left, "R"] & \mathbb{R} \arrow[l, shift left, "U"] \end{tikzcd} \qquad R \text{ is left adjoint to } U, \quad R \dashv U\ .$$

This is a symbolic version of Figure 1 with '$\dashv$' indicating a compatibility between the dialogue channels R and U through pseudoinverse-like properties. Whatever $\mathbb{R}$ does can be imitated in $\mathbb{U}$ preserving '$\leq$' in a rougher way by decoding the relevant objects in $\mathbb{U}$ to $\mathbb{R}$ using R, carrying out the operations therein, and encoding the results back to $\mathbb{U}$ using U: $a \cdot b := U\left(\mathrm{R}(a) \cdot R(b)\right)$, $\log|\,a\,| := U\left(\,\log|\,R(a)\,|\,\right)$, etc.

Now restrict the functors to monotone nondecreasing functions defined over $\mathbb{R}$ symmetric with respect to the origin,

$$\text{decode} \quad R : \mathbb{R} \to \mathbb{R} = \begin{cases} \mathbb{R}^- \ni u & \mapsto -R(-u) \\ \mathbb{R}^+ \ni u & \mapsto r \in \mathbb{R}^+ \quad \text{monotone nondecreasing} \end{cases}$$

$$\text{encode} \quad U : \mathbb{R} \to \mathbb{U} \text{ such that}$$

$$\begin{tikzcd} \mathbb{U} \arrow[r, "R"] \arrow[d, "1_{\mathbb{U}}"'] & \mathbb{R} \arrow[dl, "U"] \\ \mathbb{U} & \end{tikzcd} \qquad \text{commutes} \tag{1}$$

where $\mathbb{R}^{\pm}$ are sets of nonnegative and nonpositive reals. Then to define a function pair uniquely it suffices to specify $R^+ := R\,\big|\mathbb{R}^+$, which is decoding restricted to nonnegative reals, since U is R^{-1} on $R(\mathbb{U})$, and $R(u) = \texttt{sign}\,u \cdot R^+(|\,u\,|)$ for $u \in \mathbb{R}$.

There are a number of such FPC schemes in common use devised for various purposes:

Shorter FP : as in converting from double float to single float. The down side is that the loss of accuracy is uniform over the dynamic range providing no full cell-size accuracy where needed.

Fixed point : treats the numbers far from the origin as saturated. It often turns out that the dynamic range is too narrow.

Logarithmic : used in signal processing and other computation when the speed of multiplication and division is favored over addition and subtraction. This is a special case of FP where the mantissa is always one.

> The encoding and decoding are slow requiring logarithm, exponential, and square root functions. It also needs two sign bits, one for the number itself and the other for the exponent, which may matter when the cell is very short.

The options are narrowed down by restricting R^+ to be smooth as a function defined over $\mathbb{R}^+$. Furthermore, the last desideratum "high accuracy near zero" is interpreted to mean

$$\frac{d}{dx}R^+(x)|_{x=0} = 0 \tag{2}$$

stating that near the origin an infinitesimal $dR^+(x)$ in the decoded range may correspond to a real dx in the encoded domain. None of the FPC listed above has this property.

3 The `sqrt` encoding

By analogy to the nilsquare infinitesimal (Bell, 2001), u^2 can be taken as R^+. In this case, a 1-digit fractional encoded representation, say $x = 0.2$, has a 2-digit decoded value, $R(x) = 0.04$; see Figure 2. The dashed line is for $R(u) = u^2$ and the solid line is for $R(x)$, where x is u rounded to 1 fractional digit or $x := [10u]/10$ in which $[\ ]$ is for rounding to the nearest integer.

The FPC scheme proposed is its parameterized version

$$R^+(u; s) := s^2u^2 \tag{3}$$

where s is held constant throughout an application. This satisfies (2).

$$U^+(r; s) := \left[s^{-1}r^{\frac{1}{2}}\right] \ . \tag{4}$$

We are not aware of this FPC having been mentioned. We call this FPC the `sqrt` encoding.

This FPC is optimal in the sense that if the "simplest" nontrivial solution to (2) is accepted to be $dR^+(x)/dx = x$, (3) is the only solution.

In the rest of this section, $s := 10^{-4}$ throughout so that at $u = s$, the decoded $R^+(s) = 1$ has a four decimal digit accuracy while at $u = 0$ the accuracy is 8 decimal digits.

Table 1 shows FPC for $r = 10^n$, n integer, so $100^{\frac{1}{2}} = 10$ and $10^{\frac{1}{2}} \approx 3.162278$ alternate in the columns for u with decimal points shifted. The operator '$\circ$' is for function composition, from right to left. The table illustrates (1), that decoding and encoding back and forth does not change the original encoded integer. The `error` $:= (R \circ U)(r) - r$ and `relative error` $:=$ `error`$(r)/r$, $r \in \mathbb{R}^+$, are shown in Figures 3 and 4. Together they illustrate the trade-off between accuracy and dynamic range.

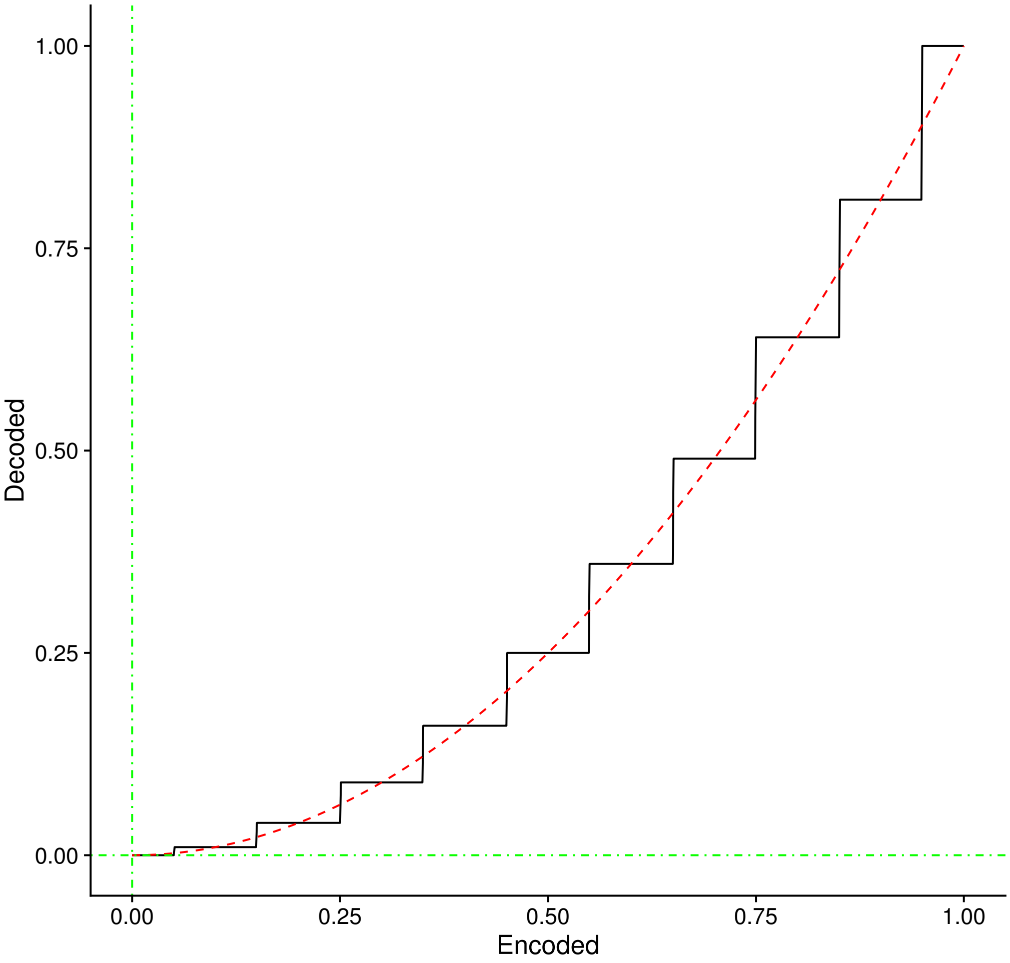

Figure 2: $R(u) = u^2$

4 Implementation

An implementation has to attend to the details left out of the model, including special symbols $\pm\infty$ and Not-a-Number, `NaN`. Currently `NaN` in float is encoded as the least integer a cell can hold. The decoding function R needs be incremented with $-M - 1 \mapsto$ `NaN` and $\pm M \mapsto \pm\infty$; U must be adjusted accordingly.

Table 1: $\mathbb{U} \xrightarrow{R} \mathbb{R} \xrightarrow{U} \mathbb{U}$ is identity on $\mathbb{U}$

r	$\overset{U}{\mapsto} u$	$\overset{R}{\mapsto} R(u) \overset{U}{\mapsto}$	$(U \circ R)(u)$
0	0	0	0
1e-09	0	0	0
1e-08	1	1e-08	1
1e-07	3	9e-08	3
1e-06	10	1e-06	10
1e-05	32	1.024e-05	32
1e-04	100	1e-04	100
0.001	316	0.00099856	316
0.01	1,000	0.01	1,000
0.1	3,162	0.09998244	3,162
1	10,000	1	10,000
10	31,623	10.00014	31,623
100	100,000	100	100,000
1000	316,228	1000.001	316,228
1e+05	3,162,278	1e+05	3,162,278
1e+06	10,000,000	1e+06	10,000,000
1e+07	31,622,777	1e+07	31,622,777
1e+08	100,000,000	1e+08	100,000,000
1e+09	316,227,766	1e+09	316,227,766
1e+10	1,000,000,000	1e+10	1,000,000,000

A program has been written for Retro 12 (Childers, 2018), a modern Forth dialect, available as a part of the language source code. Unlike the example in Section 3, the implementation is for 32-bit cells with two's complement integer. As has been mentioned in Section 1, Retro had a limited ability to deal with FP:

- operations confined to a FP stack,
- conversion to and from integer, and
- I/O.

This list has been incremented with

- encoding U and decoding R functions.

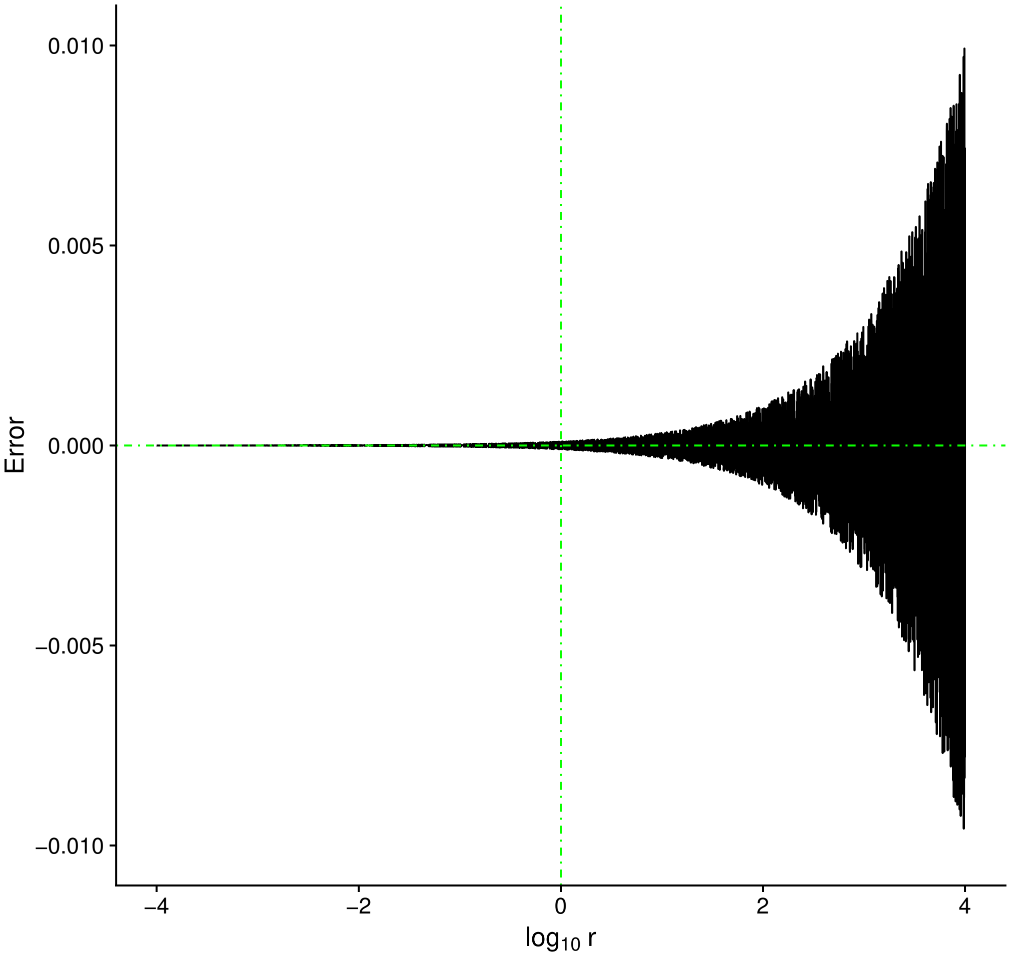

Figure 3: `error` $= (R \circ U)(r) - r$, $s = 10^{-4}$

5 Conclusion

The `sqrt` encoding of floating numbers to shorter integers has bee proposed. Its balance between accuracy and dynamic range has been illustrated in Figures 3 and 4, which is adjustable with the trade-off parameter s .

Recapitulating the FPC desiderata,

Individual : encoding and decoding depend only on the number being converted, except that the trade-off parameter s needs be specified depending on the cell size and application. For 32-bit cells it seems practical to start with a decimal s close to $2^{-32/2}$ for convenience in debugging, to be optimized once the application has been stabilized.

Fast : needs only square root, which is usually supported by FPU, in addi-

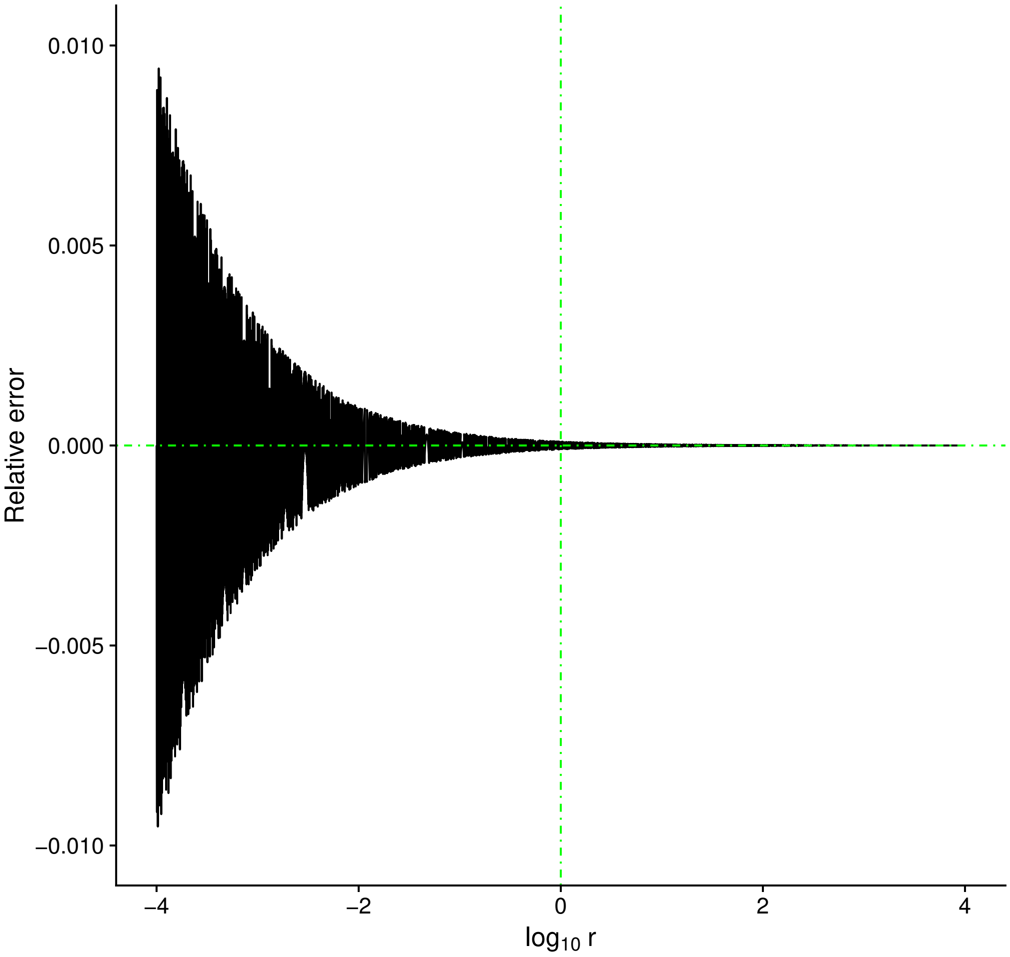

Figure 4: `relarive error` $=$ `error`$(r)/r$, $s = 10^{-4}$

tion to the basic unary and binary operations. It is useful to observe that the `sqrt` encoding is particularly well-suited for the method of least squares: to encode the square of a variable is essentially to take the absolute value of the variable itself, dispensing with the expensive square root operation.

High accuracy near zero : the accuracy is full cell size near zero deteriorating for large numbers; the relative error improves for large absolute values.

From these observations, it would seem that the trade-off parameter s should be related to the deviation from zero of the data to store.

Acknowledgments

Test code for Section 4 to build Table 1 has been written in the R language (R Core Team, 2018). The `ggplot2` (Wickham, 2009) library generated the graphs.

References

(Bell, 2001) Bell, John L. An Invitation to Smooth Infinitesimal Analysis, Mathematics Department, Instituto Superior Técnico, Lisbon, May 2001.

(Childers, 2018) Childers, Charles; Nga and Retro. 2018. `http://forthworks.com/`

(Leinster, 2016) Leinster,Tom; Basic Category Theory. arXiv:1612.09375 [math.CT], 2016. `https://arxiv.org/pdf/1612.09375.pdf`

(R Core Team, 2018) R Core Team (2018). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. `https://www.R-project.org/`

(Wickham, 2009) Wickham, H; ggplot2: Elegant Graphics for Data Analysis. Springer-Verlag New York, 2009.