Invited Paper

An Overflow/Underflow-Free Floating-Point Representation of Numbers*

SHOUICHI MATSUI** and MASAO IRI***

A new floating-point representation of numbers is proposed to cope systematically with the troubles in numerical computation due to exponent overflow/underflow in conventional floating-point representations.

The proposed representation resolves the phenomena of overflow and underflow and, at the same time, attains higher precision for numbers which are neither too large nor too small, by making mobile the boundary between the field for exponent and that for mantissa in a word.

A number system including "non-numbers" is also proposed which is closed with respect to the four arithmetic operations. The effectiveness of the proposed system is shown by numerical examples.

1. Introduction

For the numerical computation on a computer, a real number (or its approximation) is usually represented in a register of a fixed length in the form of a "floating-point" number. In the earlier days the fixed-point representation of numbers was commonly used, but, due to cumbersome problems related to "scaling", the floating-point representation, which can handle numbers over a wider range with the same relative precision at the sacrifice of a few significant digits to be allotted to the exponent, has superseded the fixed-point representation. Among a large variety of theoretically possible floating-point representation, the IBM-type (with the base 16, the mantissa represented by the sign and the magnitude, and 7-bit exponent biased by 64) is now prevalent. However, the more users experienced this type of representation of numbers for numerical computation, the more complaints arose. The exponent overflow/underflow is one of the most serious difficulties. S. Hitotumatu [7] discussed this issue. In general, as R. W. Hamming [5] points out, the range of the distribution of the exponents of numbers becomes broader and broader in the course of computation, so that the magnitudes of numbers are apt to overstride the pre-determined range, e.g. $(16^{-2^6}, 16^{2^6}) \fallingdotseq (10^{-77}, 10^{77})$. Careful scaling for avoiding this trouble would make the floating-point arithmetic less attractive, since it would invalidate the most important advantage of being scaling-free.

A floating-point number system was proposed by W. Kahan [10] to improve the undesirable properties of the IBM-type. His proposal consists basically in

(i) alloting a larger part of a word to the exponent, and

(ii) preparing "Not-A-Number (NAN)"s to guard against still possible overflow/underflow phenomena.

Other fundamental problems concerning the floating-point representation of numbers, such as the choice of the bases (=2, 4, 8, 16 etc.) [1, 2] (see also the references there) and representations different from the conventional ones [4, 6], have recently been given attention.

There is a claim [7] that the variable-length word would resolve all the difficulties, but, from the standpoints of the cost of memories and computation time, this kind of representation will not be practical except for extremely special purposes. Hence, we shall not discuss it here.

In this paper, we propose an ideal floating-point representation of real numbers under the condition that a number be represented with a fixed-length word. We discuss briefly also the characteristics of the proposed representation, its implementation, and examples of application to numerical computation.

The basic ideas of the proposed representation may be described informally as follows.

(i) When a number is represented in a fixed-length word, "bits" are allotted to the more important information with the higher priority. For example, if a number is represented by an exponent and a mantissa, a sufficient number of bits are assigned to the exponent first, and then the remaining bits are assigned to the mantissa. If the exponent is too large (or small) to be expressed in a word, then another form of representation is introduced, in which the exponent itself is represented by the "mantissa of the exponent" accompanied by the "exponent of the exponent", and a sufficient number of bits are assigned to the latter. This process will be repeated in a similar manner towards more complex representations.

(ii) A number system containing "non-number"s is established, which is closed under the four arithmetic operations, and which allows users to have complete

---

*This paper is a translation from Japanese of Transactions of IPSJ, Vol. 21, No. 4 (1980), which received an award as the 1980 winning paper of IPSJ.

**Information System Department, Economic Research Center, Central Research Institute of Electric Power Industry, Ohtemachi, Chiyoda-ku, Tokyo 100, Japan.

***Department of Mathematical Engineering and Instrumentation Physics, Faculty of Engineering, University of Tokyo, Hongo, Bunkyo-ku, Tokyo 113, Japan.

command of the numerical computation at a programming-language level (i.e. an overflow/underflow never interrupts the normal execution of computation), where the symmetry of the system with respect to the four arithmetic operations is taken account of.

The idea (i) is found, though incompletely, in R. Morris [13], and an exercise in D. E. Knuth [11] (Exercise 17 of Sec. 2.1 of Chap. 4) seems to suggest such a direction. In this paper, we shall extend the idea in full detail to propose an ideal floating-point number system. The idea (ii) which was ambiguously discussed in Kahan [10] is almost independent of (i). We shall show that, under some reasonable assumptions, the "non-number"s are determined uniquely.

2. Basic Ideas

2.1 Variable-length Exponent and Its Generalization

In a floating-point number system with the base β ($=2$, 4, 8, 16 etc.), a real number x is represented as

$$\text{(Level 0)} \qquad x=(-1)^{s_0}\times F\times\beta^{(-1)^{s_1}\times E_1}, \tag{2.1}$$

where s_0 and s_1 are either 0 or 1, E_1 is a nonnegative integer, and F is a real number which is normalized as

$$1>F\geqq 1/\beta \tag{2.2}$$

or

$$\beta>F\geqq 1. \tag{2.3}$$

For the normalization condition, the former (2.2) is widely used, but, for the reasons stated later, we adopt the latter (2.3) in the following. For any non-zero number x, the normalization condition determines its representation of the form (2.1) uniquely.

Since the number of bits to be allotted to F and E_1 is limited by the length of a word, it is obviously best to represent the exponent part E_1 exactly (if it is possible) and then to allot the remaining bits to the mantissa part F. In fact, the approximation error due to the rounding of the mantissa to a finite length is less significant than the error due to the variation of the exponent E_1 by ± 1.

By doing so, we can allocate a larger part of a word than in the conventional case to the mantissa if $|\log_\beta|x||$ is small, since the number of bits required for the exponent is small. As $|\log_\beta|x||$ becomes larger, the number of bits for E_1 increases and that for F decreases.

For those numbers, for which $|\log_\beta|x||$ is so large that E_1 may occupy most bits of a word, we are natually led to another kind of representation which we shall call the "level-1" representation:

$$\text{(Level 1)} \qquad x=(-1)^{s_0}\times F\times\beta^{(-1)^{s_1}\times E_1\times\beta^{E_2}}. \tag{2.4}$$

In this level, there is no room for the mantissa F so that we can say nothing more than that F lies between the upper and lower end of the normalization (2.3). E_1 and E_2 are determined so as to maximize the length of E_1 under the condition that E_1 and E_2 can be stored in a word and that they satisfy

$$E_1\times\beta^{E_2}\leqq E<(E_1+1)\times\beta^{E_2}, \tag{2.5}$$

where E is the exponent E_1 in the expression (2.1) when the number is expressed on level 0. We can introduce "higher levels" of representations in a similar manner. For example, for so large a number that the E_2 in (2.4) may occupy most bits of a word, the level 2 is introduced, where a number x is represented as follows:

$$\text{(Level 2)} \qquad x=(-1)^{s_0}\times F\times\beta^{(-1)^{s_1}\times E_1\times\beta^{E_2\times\beta^{E_3}}}. \tag{2.6}$$

In (2.6), F is a number which satisfies the condition (2.3), E_1 is a nonnegative integer which satisfies the condition

$$1\leqq E_1<\beta^{E_2\times\beta^{E_3}-1}, \tag{2.7}$$

and E_2 and E_3 are determined so as to maximize the length of E_2 under the condition that they can be stored in a word and they satisfy the condition

$$\beta^{E_2\times\beta^{E_3}}\leqq E<2\times\beta^{(E_2+1)\times\beta^{E_3}-1}, \tag{2.8}$$

where E is the E_1 in (2.1) when the number is expressed on level 0. Higher levels such as level 3, 4, 5, $\cdots$ are defined similarly.

For the actual implementation, additional informations, such as

(i) the level on which a number is represented,

(ii) a pointer to the boundary between the field for F and that for E_1 (in the case of the level-0 representation), or between E_i and E_{i+1} (in the case of the level-i representation with $i\geqq 1$),

are to be stored also in the same word.

2.2 Choice of the Base

The choice of the base of the number system is known to be important, and, judging from various theoretical arguments as well as experimental evidences, the best choice today should certainly be to take 2 for the base [1, 2] (see also the references there). Adopting the base 2 is advantageous also because we can suppress the first bit (i.e. the most significant bit) of the mantissa, which is always "1" in view of the normalization condition.

2.3 Sign-magnitude Notation

The representation of the mantissa as well as that of the exponent must be in the "sign-magnitude" notation, instead of "two's or one's complement" or "biased" notation, in order to realize the advantages (mentioned in §2.1 and §2.2) of the proposed representation. In passing, the sign-magnitude notation with the normalization condition (2.3) makes the number system "symmetric" with respect to the four arithmetic operations. This property of the symmetry is considered to be one of the desirable characteristics of the system [14]. Note that, for a number with the exponent of the largest absolute value, the mantissa is always equal to "1" in this system.

2.4 Rounding

The round-to-the-nearest is adopted.

2.5 "Non-numbers"

It is obvious that, within a single fixed-length word, only a finite number of real numbers can be distinguished from one another on the one hand, and on the other, that numbers larger (smaller) than the largest (smallest) representable number may possibly appear in the course of computation. Therefore, we must introduce into our number system "a number that is larger (smaller) than a certain number". It is compared to "$+\infty$" ("$-\infty$"). We shall take those "numbers which are not number in the usual sense" into our system, calling them "non-numbers". The number system containing non-numbers in addition to ordinary numbers should be closed with respect to the four arithmetic operations, so that the computation may not be interrupted by the operating system even when an overflow/underflow takes place. Then, we shall be able to describe every control of computational flow at a programming-language level, without using, e.g., the ON-statement in PL/I. Kahan also discusses such "non-number"s, "Not-A-Number (NAN)"s in his proposal [10], but they are not defined without ambiguity. We will determine such "non-number"s without ambiguity on the basis of the requirement of the symmetry of the number system. This is discussed in detail in Chap. 4.

3. Implementation of the Proposed Floating-point Number System

In practice, we can realize only a finite number of levels, so that we have to content ourselves up to a certain level. Here, we first consider the implementation of the "level 0". Higher levels may be implemented in a similar way.

3.1 Implementation of Level 0

For a real number x ($\neq 0$) expressed as in (2.1) under the normalization condition (2.3), we have the bit-strings of F and E looking like

$$F=(1.f_2f_3\cdots f_m), \tag{3.1}$$

$$E=(1e_{n-1}e_{n-2}\cdots e_1) \text{ (or } E=0). \tag{3.2}$$

Therefore, the $L=m+n$ bits

$$s_0f_2f_3\cdots f_me_{n-1}e_{n-2}\cdots e_1s_1$$

are necessary to represent x, where it is understood that $E=1$ if $n=1$ and $E=0$ (and no extra bit is needed for s_1) if $n=0$. L being fixed, n takes a value in the range from 0 to $L-1$. To hold the value of n which is the "pointer" to the boundary between the field for the mantissa and that for the exponent, $\lceil \log_2 L \rceil$ more bits are needed. Therefore, the L should be determined as the maximum integer which satisfies

$$L+\lceil \log_2 L \rceil \leqq W_L, \tag{3.3}$$

where W_L is the word length. Clearly, for any x, its representation in this form is unique, since we regard the ideal zero ("0") as a "non-number" in this system. (See Chap. 4.)

3.2 An Example of Implementation of Level 0

In this section we present an example of actual implementation of level 0. First, we assume that a word consists of 64 bits. Therefore, we have $L=58$ and $\lceil \log_2 58 \rceil = 6$. The data structure for the level 0 is shown in Fig. 1. The length of the mantissa (without the sign bit) varies from 58 to 1 (including the suppressed first bit), and consequently, that of the exponent (with the sign bit) Varies from 0 to 57 (including the most significant bit, which is suppressed, when $B=n\neq 0$). Therefore, the numbers expressible on level 0 range widely, i.e. from about $10^{-10^{16}}$ to about $10^{10^{16}}$.

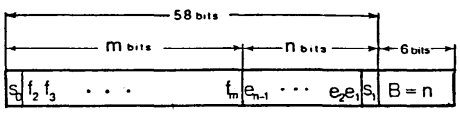
The values of B from 58 to 63 are meaningless, but they can be used to indicate (i) non-numbers, (ii) higher-level representations, etc.

The printing format of a level-0 number may be decided on the same principle as the partition of a word into the mantissa and the exponent part was based on. A specified field length of printing is divided into two parts, one for the exponent and the other for the mantissa. First, sufficient digits in the field are allocated to the exponent part, and then the remaining digits are allocated to the mantissa which is normalized in the form (X.XXX$\cdots$). (See Chap. 6 for an example.)

For the input formats, we allow all the formats which have been used in conventional computer systems.

4. A Floating-point Number System Including Non-numbers

A number system for numerical computation should desirably be closed at least with respect to the four arithmetic operations, so that, even if "an illegal pair of operands" appears, it should not disturb the normal progression of a programme. In general, a number represented within a fixed-length word is to be regarded as an interval rather than a point on the real line, regardless whether the width of the interval is explicitly expressed or not. Let us begin with the set of nonzero real numbers $R^+\cup R^-$, where



mantissa part $(-1)^{s_0}F=(-1)^{s_0}\times(1.f_2f_3\cdots f_m)_2$
exponent part $(-1)^{s_1}E=(-1)^{s_1}\times(1e_{n-1}\cdots e_2e_1)_2$
$B=n$ points to the boundary between the mantissa part and the exponent part.
($B=0, 1, \cdots, 57$; $E=0$ when $B=0$)

Fig. 1 Data structure for a floating-point number in the proposed representation (level 0)

$$R^+ = \{\text{positive real numbers}\}$$
and (4.1)
$$R^- = \{\text{negative real numbers}\}.$$

Only part of the elements of $R^+ \cup R^-$ can be represented in the form shown in Chap. 3. For the symmetry and closedness of the number system, the following 7 types of intervals (as well as those types of intervals which are defined with $<$ in place of $\leqq$) are necessary and sufficient:

(i) $\{x|a \leqq x \leqq b;\ a, b \in R^+\}$, $\{x|a \leqq x \leqq b;\ a, b \in R^-\}$,

(ii) $\{x|a \leqq x;\ a \in R^+\}$, $\{x|x \leqq a;\ a \in R^-\}$;

(iii) $\{x|0 < x \leqq a;\ a \in R^+\}$, $\{x|a \leqq x < 0;\ a \in R^-\}$;

(iv) $\{x|x \leqq a \text{ or } b \leqq x;\ a \in R^-, b \in R^+\}$;

(v) $\{x|a \leqq x \leqq b;\ a \in R^-, b \in R^+\}$;

(vi) R^+, R^-;

(vii) $R(=R^+ \cup R^- \cup \{0\})$.

In fact, starting from intervals of the basic type (i), all the other types of intervals can be produced by the four arithmetic operations. The following symbols are used to denote the above 7 types of intervals, respectively.

(i) $+\text{num}, -\text{num}$;

(ii) $+\infty, -\infty$;

(iii) $+0, -0$;

(iv) ∞;

(v) 0;

(vi) $+?, -?$;

(vii) $?$.

The intervals of types (ii)$\sim$(vii) are called "non-numbers", which are schematically shown in Fig. 2.

We regard a number which is representable in the form shown in Chap. 3 as the representative point of an interval of type $\pm$num, and assume the result of an arithmetic operation on two numbers of that kind to be one of the numbers and non-numbers of the above 7 kinds, as follows.

a. If the result is representable in the form shown in Chap. 3, then round it to an element of $\pm$num, if necessary.
b. If the absolute value of the result is too large to be represented in the form of Chap. 3, then denote it by $\pm\infty$.
c. If the absolute value of the result is too small to be represented in the form of Chap. 3, then,
   c1. if the mantissa preserves information on the sign, then denote it by ± 0;
   c2. if the mantissa carries no information on the sign, then denote it by 0.

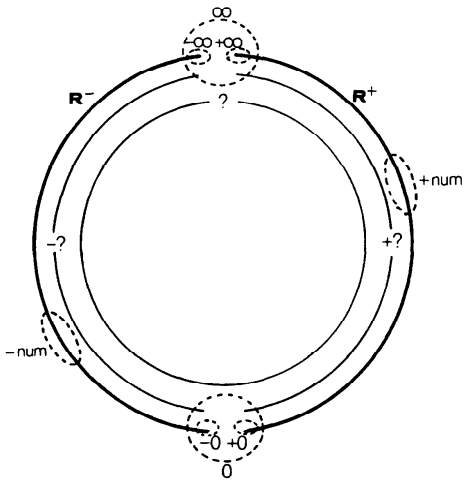


Fig. 2 Schematic diagram of the proposed number system including non-numbers.

The tables for the four arithmetic operations among numbers and non-numbers are given in Tables 1, 2, 3 and 4. These tables will explain how all the 7 types of intervals actually appear, and why those non-numbers are necessary and sufficient for the number system to be closed.

The order among numbers and non-numbers is defined in terms of the sign carried by the result of the subtraction operation as follows:

$X > Y$ iff the sign of $X - Y$ is "+";

$X < Y$ iff the sign of $X - Y$ is "−";

Table 1 Addition table for the number system including non-numbers.

op1 \ op2	+num	−num	+∞	−∞	+0	−0	+?	−?	∞	?	0
+num	+num +∞	±num ±0, 0	+∞	−∞	+num	+num	+?	?	∞	?	+num
−num	±num ±0, 0	−num −∞	+∞	−∞	−num	−num	?	−?	∞	?	−num
+∞	+∞	+∞	+∞	?	+∞	+∞	+∞	?	?	?	+∞
−∞	−∞	−∞	?	−∞	−∞	−∞	?	−∞	?	?	−∞
+0	+num	−num	+∞	−∞	+0	0	+?	?	∞	?	0
−0	+num	−num	+∞	−∞	0	−0	?	−?	∞	?	0
+?	+?	?	+∞	?	+?	?	+?	?	?	?	?
−?	?	−?	?	−∞	?	−?	?	−?	?	?	?
∞	∞	∞	?	?	∞	∞	?	?	?	?	∞
?	?	?	?	?	?	?	?	?	?	?	?
0	+num	−num	+∞	−∞	0	0	?	?	∞	?	0

Table 2 Subtraction table for the number system including non-numbers

op1 \ op2	+num	−num	+∞	−∞	+0	−0	+?	−?	∞	?	0
+num	±num ±0, 0	+num +∞	−∞	+∞	+num	+num	?	+?	∞	?	+num
−num	−num −∞	±num ±0, 0	−∞	+∞	−num	−num	−?	?	∞	?	−num
+∞	+∞	+∞	?	+∞	+∞	+∞	?	+∞	?	?	+∞
−∞	−∞	−∞	−∞	?	−∞	−∞	−∞	?	?	?	−∞
+0	−num	+num	−∞	+∞	0	+0	?	+?	∞	?	0
−0	−num	+num	−∞	+∞	−0	0	−?	?	∞	?	0
+?	?	+?	?	+∞	?	+?	?	+?	?	?	?
−?	−?	?	−∞	?	−?	?	−?	?	?	?	?
∞	∞	∞	?	?	∞	∞	?	?	?	?	∞
?	?	?	?	?	?	?	?	?	?	?	?
0	−num	+num	−∞	+∞	0	0	?	?	∞	?	0

Table 3 Multiplication table for the number system including non-numbers

op1 \ op2	+num	−num	+∞	−∞	+0	−0	+?	−?	∞	?	0
+num	+num +∞, +0	−num −∞, −0	+∞	−∞	+0	−0	+?	−?	∞	?	0
−num	−num −∞, −0	+num +∞, +0	−∞	+∞	−0	+0	−?	+?	∞	?	0
+∞	+∞	−∞	+∞	−∞	+?	−?	+?	−?	∞	?	?
−∞	−∞	+∞	−∞	+∞	−?	+?	−?	+?	∞	?	?
+0	+0	−0	+?	−?	+0	−0	+?	−?	?	?	0
−0	−0	+0	−?	+?	−0	+0	−?	+?	?	?	0
+?	+?	−?	+?	−?	+?	−?	+?	−?	?	?	?
−?	−?	+?	−?	+?	−?	+?	−?	+?	?	?	?
∞	∞	∞	∞	∞	?	?	?	?	∞	?	?
?	?	?	?	?	?	?	?	?	?	?	?
0	0	0	?	?	0	0	?	?	?	?	0

Table 4 Division table for the number system including non-numbers

op1 \ op2	+num	−num	+∞	−∞	+0	−0	+?	−?	∞	?	0
+num	+num +∞, +0	−num −∞, −0	+0	−0	+∞	−∞	+?	−?	0	?	∞
−num	−num −∞, −0	+num +∞, +0	−0	+0	−∞	+∞	−?	+?	0	?	∞
+∞	+∞	−∞	+?	−?	+∞	−∞	+?	−?	?	?	∞
−∞	−∞	+∞	−?	+?	−∞	+∞	−?	+?	?	?	∞
+0	+0	−0	+0	−0	+?	−?	+?	−?	0	?	?
−0	−0	+0	−0	+0	−?	+?	−?	+?	0	?	?
+?	+?	−?	+?	−?	+?	−?	+?	−?	?	?	?
−?	−?	+?	−?	+?	−?	+?	−?	+?	?	?	?
∞	∞	∞	?	?	∞	∞	?	?	?	?	∞
?	?	?	?	?	?	?	?	?	?	?	?
0	0	0	0	0	?	?	?	?	?	?	?

$X = Y$ iff X and Y have the same representation, i.e. they have the same bit-pattern.

With this definition, we have, for example,

$$-\infty < -\text{num} < -0 < +0 < +\text{num} < +\infty,$$

$$-\text{num} < 0 < +\text{num},$$

if $x < y$ and $y < z$, then $x < z$.

However, it is not always true that

if $x = y$ and $u < v$, then $x + u < y + v$.

5. Representation Error in the Proposed System

Usually, for a real number x, the representative point x^* of the interval containing x is obtained by rounding the next lower digit of the least significant representable digit of the mantissa of x. This convention is adopted for the level-0 numbers in our system, and the representative points of the numbers of higher levels are defined similarly.

The "representation error" of a number x is, then,

defined by

$$e_{rep}(x)=\max_{y}\left\{\frac{|x^*-y|}{|y|}\,\middle|\, y \text{ has the same representative point } x^* \text{ as } x\right\}. \tag{5.1}$$

The "unit of rounding" of a number x is defined by

$$u(x)=\max_{y}\{e_{rep}(y)|y \text{ has the same exponent as } x\}. \tag{5.2}$$

As is easily seen, for a floating-point number x with the base β and the L-digit mantissa, we have

$$u(x)=\frac{1}{2}\beta^{1-L} \tag{5.3}$$

in the case where the round-to-the-nearest is adopted.

For the "non-number"s, it may be assumed that $e_{rep}(x)$ is undefined, or defined to be ∞. In this respect, it is natural to take "0" and "± 0" for "non-number"s, rather than numbers.

The "representation error" as well as the "unit of rounding" is a good measure in terms of which to compare different manners of floating-point representation. We shall consider here the following four typical floating-point number systems for comparison. For convenience' sake, the comparison will be made using the 64-bit word.

(i) IBM-type:———The base is 16. The mantissa has 56 bits. The rounding is "chopping". The exponent has 7 bits ("biased" notation). The sign of the mantissa occupies 1 bit. (This is the so-called DOUBLE-PRECISION number).

(ii) Kahan-type:———The base is 2. The mantissa has 52 bits (substantially, 53 bits since the first bit is suppressed). The rounding is the round-to-the-nearest-to-even. The exponent has 11 bits ("biased" notation). The sign of the mantissa occupies 1 bit. ("DOUBLE" [8].) "Denormalized numbers" are also used to resist the underflow to some extent.

(iii) Morris-type:———The adaptation of his original proposal with the 36-bit word for the 64-bit word. The base is 2. The mantissa and the exponent use 61 bits. There is a pointer, G (3 bits), to the boundary between the field of the mantissa and that of the exponent. The rounding is the round-to-the-nearest. The length of the exponent field is specified by G as

(i) $G+1$ (Morris' type (i)),

or

(ii) $G+4$ (Morris' type (ii)).

(iv) Our system:———See Chap. 3 for the details. (We mainly consider the level 0. The rounding is the round-to-the-nearest as in Chap. 2.)

The normalization condition is (2.2) for (i) and (iii), and (2.3) for (ii) and (iv).

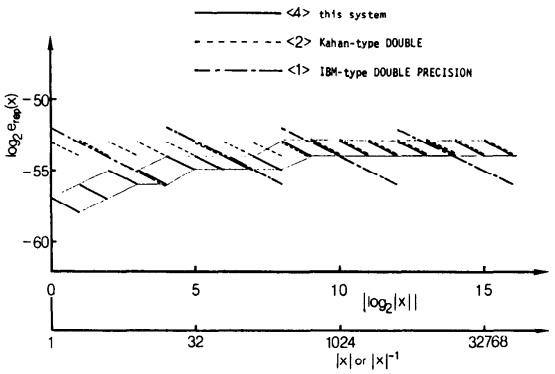


Fig. 3 Representation error $e_{rep}(x)$ of various systems.

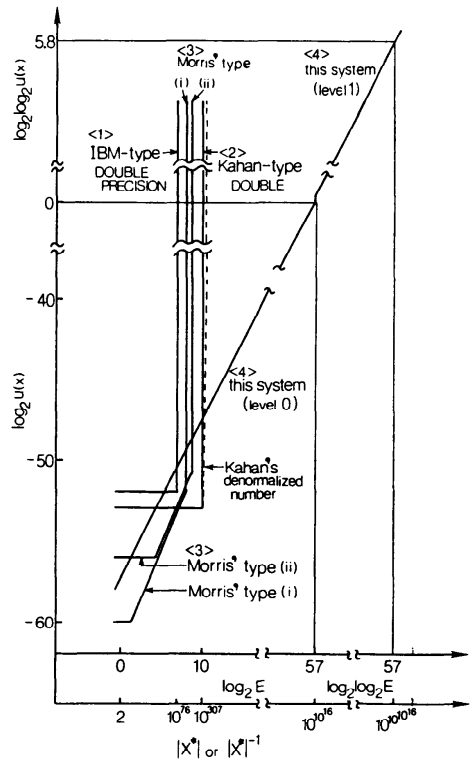


Fig. 4 Units of rounding $u(x)$ for various systems.

In Fig. 3 the representation error $e_{rep}(x)$ for the numbers whose absolute value is neither too large nor too small is shown. In Fig. 4 the unit of rounding $u(x)$ is shown for a wider range of numbers.

As is seen in Fig. 3 and Fig. 4, the proposed representation of numbers has no "discontinuity" in precision, but the representation error "continuously" depends on the magnitude of a number. Furthermore, higher precision than the IBM-type or the Kahan-type is realized for numbers of moderate magnitude.

6. Numerical Examples Computed by Means of the Proposed System

6.1 Experimental Programmes

The proposed floating-point number system is implemented in the manner shown in Chap. 3 as a system of subroutines and functions written in FORTRAN on the HITAC 8800/8700 Computer Complex of the Computer Centre of the University of Tokyo. The experimental system of programmes consists of

(1) the subroutines to perform the four arithmetic operations (approximately 300 lines in total);
(2) the function subprogrammes to perform the input/output of numbers (approximately 200 lines in total);
(3) the function subprogrammes to perform the type conversion of the data in FORTRAN into those in our system and vice versa (approximately 200 lines in total);
(4) the function subprogrammes such as square root, exponential, and logarithm (approximately 200 lines in total);
(5) the auxiliary subroutines for comparison, rounding and pack/unpack operation of the mantissa and the exponent (approximately 300 lines in total).

All the non-numbers mentioned in Chap. 4 can be treated in the current version.

These subroutines and functions are merely for experimental use, so that they are not fast; e.g., for usual numerical computations, they run 600 through 1,000 times as slowly as the conventional FORTRAN statements on the HITAC system. A considerable speed-up, however, can be achieved by coding those subprogrammes in an assembly language. Moreover, it is theoretically possible that the proposed system runs as fast as the IBM-type or the Kahan-type number system if special-purpose arithmetic units are incorporated in the CPU.

6.2 Numerical Examples of Practical Computations

In this section, we show the effectiveness of the proposed number system by applying it to practical computational problems. There are so many difficult problems to solve by a simple program because both very large and small numbers appear in the course of computation. For example, the Graeffe method for algebraic equations is difficult to implement because coefficients may become very large or small quickly, even if the zeroes are neither very large nor very small. In order to solve these problems by a computer with a conventional floating-point number system such as that of the IBM-type, a highly sophisticated program equipped with a number of tricks to protect against overflow/underflow will be required.

The proposed system enables the users to solve those difficult problems by simple programs, and the solutions attained are generally no less accurate than those attained by complicated (or sophisticated) programs with conventional floating-point number systems.

6.2.1 Graeffe Method

Let us consider the solution of algebraic equations by the Graeffe method [18], where both very large and very small numbers often appear in the course of computation. For simplicity, we restrict our attention to the polynomials with all the zeroes positive real.

Let α_i $(i=1,\cdots,n)$ be the zeroes of a given polynomial

$$P(x)=x^n-a_{n-1}x^{n-1}+\cdots+(-1)^{n-1}a_1x+(-1)^n a_0,$$

and let the polynomial with zeroes $\alpha_i^{2^\nu}$ $(i=1,\cdots,n)$ be

$$P^{(\nu)}(x)=x^n-a_{n-1}^{(\nu)}x^{n-1}+\cdots+(-1)^{n-1}a_1^{(\nu)}x+(-1)^n a_0^{(\nu)}. \tag{6.1}$$

Then $a_i^{(\nu)}$ $(i=0,\cdots,n-1)$ satisfy the simple recurrence relation:

$$a_k^{(0)}=a_k, \tag{6.2}$$
$$a_k^{(\nu+1)}=\sum_{i+j=k}(-1)^{k+i}a_i^{(\nu)}a_j^{(\nu)} \quad (\nu=0,1,\cdots),$$

and the ν-th approximations to α_i are given by

$$\alpha_i \doteqdot (a_{i-1}^{(\nu)}/a_i^{(\nu)})^{1/2^\nu}. \tag{6.3}$$

Fig. 5 shows the iteration process using our system for the polynomial

$$\begin{aligned}P_1(x)&=x^8-11x^7+45.35x^6-88.55x^5+86.7542x^4\\&\quad-43.274x^3+10.984x^2-1.32x+0.0576\\&=(x-0.1)(x-0.2)(x-0.3)(x-0.4)(x-1)\\&\quad\times(x-2)(x-3)(x-4),\end{aligned} \tag{6.4}$$

and Fig. 6 shows that for the polynomial

$$\begin{aligned}&P_2(x)\\&=x^4-10.43857593020614x^3+40.58740567587410x^2\\&\quad-69.60408570545396x^2+44.36715614906059\\&=(x-2)(x-e)(x-\sqrt{7.4})(x-3)\\&\quad(e=2.718281828\cdots,\ \sqrt{7.4}=2.7202941017\cdots).\end{aligned} \tag{6.5}$$

In Fig. 5, all the zeroes are determined to a sufficient precision after 7 iterations ($\nu=7$). Note that the subsequent iterations do not deteriorate the accuracy. Iterations over the 6-th is impossible with the IBM-type representation of numbers.

In Fig. 6, it is seen that, due to the existence of nearly multiple zeroes, more iterations are required to get good approximations for zeroes, and that the final approximations are not very accurate. But, in our system, all the zeroes are determined after 16 iterations ($\nu=16$) with good accuracy, up to about 10 decimal digits. The IBM-type cannot proceed farther than the 6-th iteration, at which approximations are very poor—up to about 2 decimal places in the worst case. Neither can the Kahan-type proceed farther than the 7-th iteration at which it gives no better results than the IBM-type.

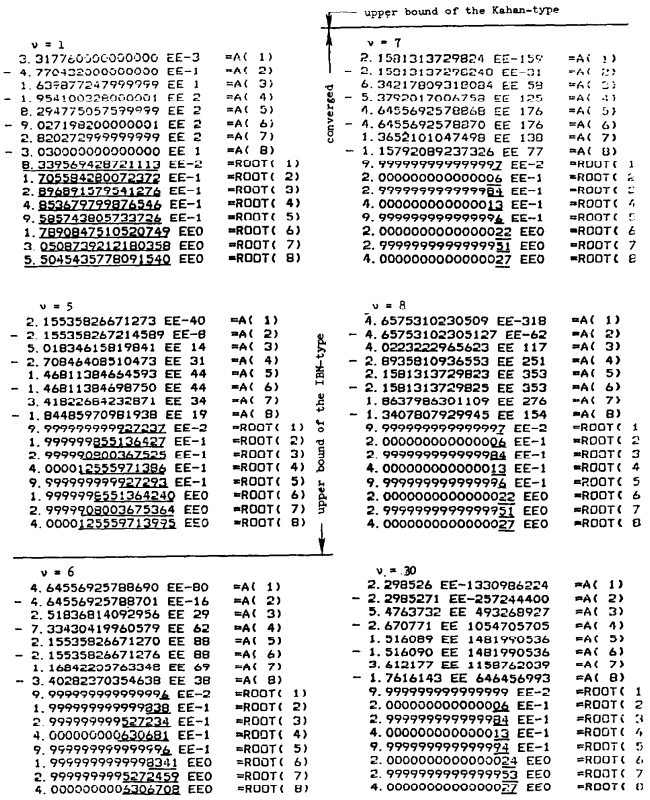


Fig. 5 Iteration by the Graeffe method for $P_1(x)$.

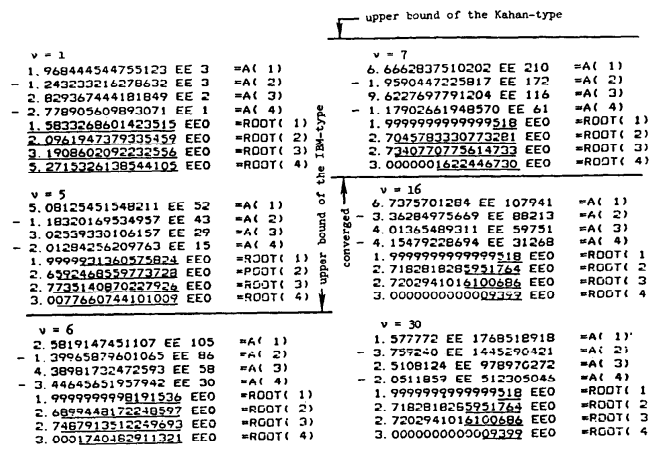


Fig. 6 Iteration by the Graeffe method for $P_2(x)$.

In Fig. 7, the maximum relative errors of computed zeroes together with the maximum magnitudes of the coefficients are plotted against the iteration numbers.

6.2.2 Binomial Probability

Let us consider the problem of calculating the binomial probability

$$x_k = \binom{N}{k} p^k q^{N-k} \quad (p+q=1) \tag{6.6}$$

for a large N, say $N=2{,}000$ [16]. An N as large as 2,000 usually appears in practical problems of inventory control.

A simple program such as that shown in Fig. 8 will

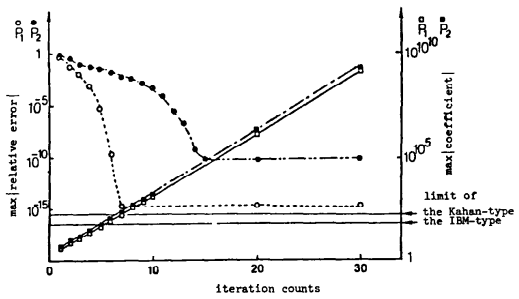


Fig. 7 Iteration by the Graeffe method for $P_1(x)$ and $P_2(x)$.

```
program 1
function Binomial(N,k:integer;p,q:real):real;
real: x; integer: i;
begin x:=1.0;
  for i:=1 to k do x:=x*(N-k+i)/i*p;
  for i:=1 to N-k do x:=x*q;
  return x
end
```

Fig. 8 A simple program to calculate binomial probability.

```
1.148130695274 EE-1398  = X(  0)
9.185045562194 EE-1395  = X(  1)
3.672181215765 EE-1391  = X(  2)
9.782690758799 EE-1388  = X(  3)

2.224107084496770 EE-2  = X(1599)
2.229667352208012 EE-2  = X(1600)
2.228274680532679 EE-2  = X(1601)
2.219929082478250 EE-2  = X(1602)
```

Fig. 9 Example of binomial probability computed by our system (N=2,000, p=0.8, q=0.2).

certainly fail because of overflow/underflow on a conventional system.

On a conventional system, a more sophisticated program which incorporates elaborate scaling techniques will be needed to get an exact answer avoiding overflow/underflow (see [16] for an example of complete program of that kind).

If we use the proposed floating-point number system, any simple program will work well, and even faster than the sophisticated one. Examples of binomial probabilities computed by the proposed system are shown in Fig. 9.

6.2.3 Divergence-convergence Boundary of a Quadratic Transformation

Let us consider the problem of calculating the divergence-convergence boundary of the quadratic transformation

$$x'^{\kappa}=\sum_{\lambda=1}^{n}\sum_{\mu=1}^{n}P_{\lambda\mu}^{\kappa}x^{\lambda}x^{\mu}\quad(\kappa=1,2,\cdots,n). \tag{6.7}$$

This kind of transformation characterizes the asymptotic behaviour of the Newton-Raphson process near the solution (see [3] for details).

For $n=2$, the divergence-convergence boundary for the fixed direction θ is calculated by the algorithm shown in Fig. 10 [3]. The restart step, i.e., step 3.1, of the algorithm is used for avoiding overflow/underflow, and the presence of this step increases the total computing time.

```
1°   r0 ← 1
2°   x0 ← r0cos θ ;  y0 ← r0sin θ ;  m ← 1
3°   xm ← f1(xm-1,ym-1)
     ym ← f2(xm-1,ym-1)
     rm ← √(xm² + ym²)
3.1° if rm > 10^30 or rm < 10^-30
       then r0 ← r0/rm^(2^-m) ; goto 2°
3.2° if m=15 then goto 4°
       else m ← m+1; goto 3°
4°   r*(divergence-convergence boundary) ← r0/r15^(2^-15)
```

Fig. 10 Procedure to calculate the divergence-convergence boundary of the quadratic transformation (n=2).

Table 5 Comparison of the performance between the IBM-type and our system for a quadratic transformation.

p_1	p_2	IBM-type		Our System	
		#restart	#total iterations	#restart	#total iterations
0.0	0.0	0	75,000	0	75,000
0.0	0.1	4,742	134,706	0	75,000
0.0	1.2	7,586	155,816	0	75,000
0.0	10.0	10,000	169,332	0	75,000
3.7	2.5	10,000	180,666	0	75,000

Since the proposed system can handle very wide ranges of numbers, no restart step is required, so that the total computing time is not increased.

In Table 5, comparison of the performances of the IBM-type number system and the proposed number system is shown for the quadratic transformation

$$\begin{aligned}x'&=f^1(x,y)=-2xy+2/3x(p_1x+p_2y),\\ y'&=f^2(x,y)=x^2+y^2+2/3y(p_1x+p_2y).\end{aligned}\tag{6.8}$$

The divergence-convergence boundary of the transformation (6.8) is calculated for $\theta=\theta_i=2\pi i/10{,}000$ ($i=1,\cdots,5{,}000$) to sufficient accuracy with the same stopping rule on the two systems. Table 5 indicates that the IBM-type system requires two restart steps for one θ_i on the average, while our system requires no restart step and the total iteration is nearly halved compared with the IBM-type.

6.2.4 6-j Symbols (Racah Symbols)

Many problems in quantum mechanics are formulated in terms of angular momenta and their combinations. 6-j symbols (Racah symbols), as defined below, are the coefficients used for combining angular momenta [9].

Definition. 6-j symbols $\begin{Bmatrix}j_1 j_2 j_3\\ l_1 l_2 l_3\end{Bmatrix}$:

$$\begin{Bmatrix}j_1 j_2 j_3\\ l_1 l_2 l_3\end{Bmatrix}\equiv\Delta(j_1j_2j_3)\Delta(j_1l_2l_3)\Delta(l_1j_2l_3)\Delta(l_1l_2j_3)w\begin{Bmatrix}j_1 j_2 j_3\\ l_1 l_2 l_3\end{Bmatrix},$$

$$\Delta(abc)\equiv\sqrt{\frac{(a+b-c)!(a-b+c)!(-a+b+c)!}{(a+b+c+1)!}},$$

$$w\begin{Bmatrix} j_1 j_2 j_3 \\ l_1 l_2 l_3 \end{Bmatrix} \equiv \sum_z \frac{(-1)^z (z+1)!}{F(z, j_1, j_2, j_3, l_1, l_2, l_3)},$$

$$F(z, j_1, j_2, j_3, l_1, l_2, l_3)$$
$$\equiv (z-j_1-j_2-j_3)!(z-j_1-l_2-l_3)!(z-l_1-j_2-l_3)!$$
$$\times (z-l_1-l_2-j_3)!(j_1+j_2+l_1+l_2-z)!$$
$$\times (j_2+j_3+l_2+l_3-z)!(j_3+j_1+l_3+l_1-z)!,$$

where j_*'s and l_*'s are nonnegative half integers such that the three arguments appearing in one and the same $\Delta(a\,b\,c)$ may satisfy the triangular inequality, and the summation $\sum_z$ is taken for all integers such that the arguments of factorials may be nonnegative.

The calculation of the 6-j symbols has been tried by many researchers. There are also some explicit formulas for the 6-j symbols with relatively small momenta. Numerical tables are available for the angular momenta not exceeding eight [9, 15].

For nuclear physics, the 6-j symbols with small arguments are sufficient, and they are easy to compute directly from the definition. But, for some problems such as ion-beam scattering, we have to combine rather large angular momenta. In that case, a simple program does not work due to overflow and underflow if we use a conventional floating-point system. The proposed system enables us to run a simple program without any trick to get satisfactory results.

In Fig. 11, a few examples are shown of the computation of 6-j symbols with rather large angular momenta by means of the proposed system.

6.2.5 Newton-Raphson Iteration

The Newton-Raphson iteration for the solution of nonlinear equations is often embarrased by overflow/underflow. As is well known, this iteration process converges quadratically to a solution so long as the initial approximation is sufficiently close to the solution, and, in such a case, no overflow/underflow occurs. However, if the initial approximation is far from the solution, the convergence is slow, or, sometimes, a big jump takes place, which may cause overflow/underflow. However, the proposed system will work well even in those cases. Some discussions are made in [12].

$$\begin{Bmatrix} 10 & 10 & 10 \\ 10 & 10 & 10 \end{Bmatrix} = -2.91918678060921\underline{7}\ \text{EE-3}$$

$$\begin{Bmatrix} 20 & 20 & 20 \\ 20 & 20 & 20 \end{Bmatrix} = -5.02940645686852\underline{2}\ \text{EE-3}$$

$$\begin{Bmatrix} 30 & 30 & 30 \\ 30 & 30 & 30 \end{Bmatrix} = 4.10235321596938\underline{0}\ \text{EE-4}$$

$$\begin{Bmatrix} 40 & 40 & 40 \\ 40 & 40 & 40 \end{Bmatrix} = 1.82830697216527\underline{6}\ \text{EE-3}$$

$$\begin{Bmatrix} 50 & 50 & 50 \\ 50 & 50 & 50 \end{Bmatrix} = -1.12137431660331\underline{0}\ \text{EE-4}$$

$$\begin{Bmatrix} 60 & 60 & 60 \\ 60 & 60 & 60 \end{Bmatrix} = -1.00663625630632\underline{2}\ \text{EE-3}$$

Fig. 11 Examples of 6-j symbols for large angular momenta.

7. Concluding Remarks

In this paper, we proposed basic ideas of a new floating-point number system, compared it with the conventional ones, and gave examples of its implementation and numerical computations by means of it. The following problems are to be further investigated.

(i) Hardware implementation:———The advantages of our proposal will be more clearly proved by implementing it by hardware arithmetic circuits. The design and the actual implementation are being done. Theoretically, it is possible to realize as fast a speed as that realized by the conventional floating-point arithmetic circuits.

(ii) Rounding-error analysis:———We have discussed only the approximation error occurring when we represent a real number in a fixed-length word. The well-known analysis on the basis of the "fixed-length mantissa" [18] does not apply to our system. An analysis technique must be developed which can deal not only with numbers but also with non-numbers, and which is also capable of considering propagation, accumulation, and magnification of rounding errors in the process of numerical computation.

(iii) Practical implementation of higher levels and non-numbers:———Input/output format of higher-level numbers and non-numbers must be standardized. How many levels to consider will also be investigated from the theoretical as well as practical standpoint.

Acknowledgement

The authors are grateful to Professor Sigeiti Moriguti of the University of Electro-Telecommunication, Professor Emeritus of the University of Tokyo, and Professor Hideo Aiso of Keio University for their valuable suggestions and comments given to the authors at informal meetings. The authors are grateful also to the members of their research group for helpful discussions. Those suggestions, comments and discussions were very helpful to the authors in improving the present paper.

Reference

1. BRENT, R. P. On the Precision Attainable with Various Floating-Point Number System. *IEEE Trans. on Computers*, **C-22**, 6 (1973), 601–607.
2. CODY, W. J., Jr. Static and Dynamic Numerical Characteristics of Floating-Point Arithmetic. *IEEE Trans. on Computers*, **C-22**, 6 (1973), 598–601.
3. DATE, T. Properties of Divergence-Convergence Boundary of Quadratic Transformations (in Japanese). *Journal of Information Processing Society of Japan*, **19**, 6 (1978), 507–513.
4. EDGER, A. D. and LEE, S. C. FOCUS: Microcomputer Number System. *Comm. ACM*, **22**, 3 (1979), 166–177.
5. HAMMING, R. W. On the Distribution of Numbers. *The Bell System Technical Journal*, **40**, 8 (1970), 1609–1625.
6. HEHNER, E. C. R. and HORSPOOL, R. N. S. A New Representation of the Rational Numbers for Fast Easy Arithmetic. *SIAM Journal on Computing*, **8**, 2 (1979), 124–134.

7. HITOTUMATU, S. Proposal of a New Standard Floating Point Arithmetic System (in Japanese). *Journal of Information Processing Society of Japan*, **20**, 9 (1979), 793–797.
8. IDA, T. and GOTO, E. Overflow Free and Variable Precision Computing in FLATS. *Journal of Information Processing*, **1**, 3 (1978), 140–142.
9. JUDD, B. R. *Angular Momentum Theory for Diatomic Molecules.* Academic Press, New York, N.Y., 1975.
10. KAHAN, W. and PALMER, J. On a Proposed Floating-Point Standard. *ACM SIGNUM Newsletter*, Special Issue (Oct. 1979), 13–21.
11. KNUTH, D. E. *The Art of Computer Programming*, 2: *Seminumerical Algorithms.* Addison-Wesley, Reading, Mass., 1969.
12. MATSUI, S. *On Numerical Computation Oriented Floating-Point Number Systems* (in Japanese). Master Thesis, University of Tokyo, March 1981.
13. MORRIS, R. Tapered Floating Point: A New Floating-Point Representation. *IEEE Trans. on Computers*, **C-20**, 6 (1973), 1678–1679.
14. REINSH, C. H. Principle and Preference for Computer Arithmetic. *ACM SIGNUM Newsletter*, **14**, 1 (1979), 12–27.
15. ROTENBERG, et al. *The 3-j and 6-j Symbols.* Technology Press, MIT, Cambridge, Mass., 1959.
16. STERBENZ, P. H. *Floating-Point Computation.* Prentice-Hall, Englewood Cliffs, N.J., 1974.
17. SWARTZLANDER, E. E. and ALEXOPOULOS, A. G. The Sign/Logarithm Number System. *IEEE Trans. on Computers*, **C-24**, 12 (1975), 1238–1243.
18. WILKINSON, J. H. *Rounding Errors in Algebraic Processes.* Prentice-Hall, Englewood Cliffs, N.J., 1963.