Estimates of Rounding Errors with Fast Automatic Differentiation and Interval Analysis

KOICHI KUBOTA* and MASAO IRI**

We propose an algorithm for calculating rigorous upper bounds of the absolute values of the rounding errors incurred in the computed values of functions. We also prove that, under some suitable conditions, the upper bounds become asymptotically sharp as the machine epsilon approaches zero. The proposed method is a combination of the technique of Fast Automatic Differentiation and interval operations for real intervals with both upper and lower edges representable as floating-point numbers.

The method is useful to guarantee the quality of numerical results in the sense that we can determine an interval containing the exact value of the function based on the value computed in finite precision and on the estimate obtained by the method. We also illustrate the efficiency and the practicalness of the proposed method by numerical experiments in comparison with the conventional interval analysis.

1. Introduction

By means of Fast Automatic Differentiation [3, 5, 6, 7, 8, 11], we can practically estimate rounding errors that occur in the computed values of functions. Indeed, two methods of estimating rounding errors called "Absolute Bounds" and "Probabilistic Estimates," have been proposed and reportedly give good approximations to rounding errors [7, 11]. But we cannot theoretically rigorously guarantee that these methods will give a range containing the *exact* function value (namely, the value that would have been obtained without rounding errors). To obtain such a precise range for the function value, we may resort to a kind of interval arithmetic by replacing arithmetic operations to be executed in computation of the functions with the corresponding machine interval operations. However, for large complicated functions, the width of the interval thus obtained as the final result may be unacceptably wide. In this paper, we propose an algorithm for calculating estimates that give rigorous and sharp upper bounds for the absolute values of rounding errors and that are proved to be "optimal" in a sense defined later. We also compare the estimates obtained by means of the proposed algorithm with those obtained by machine interval operations and with absolute bounds.

The proposed algorithm is a combination of the technique of Fast Automatic Differentiation and Machine Interval Operations. We assume that there are no errors in input data, and investigate only the errors generated and accumulated in the course of computation. (It is straightforward to extend the discussion to cases in which the input data are contaminated with errors.) Our standpoint is as follows: "Each operation in computation is performed at the highest precision possible on the available machine, so we cannot know the exact value of rounding error, or even its sign, but only the upper bound of the absolute value of the rounding error generated on the operation." This assumption will be accepted as plausible when rounding errors are rigorously discussed.

The basic principle of our algorithm is the "mean value theorem" in differential calculus. In interval analysis of a function with many variables, E.R. Hansen has already proposed and used a method based on the mean value theorem, but his method requires a computational time proportional to the product of the number of variables and the time required to compute the function itself [4, 14]. Yu. V. Matiyasevich proposed another method, an interval analysis with mean value theorem that makes use of the idea of Fast Differentiation, and showed that the computational time of the method was independent of the number of variables [12]. Our method proposed in this paper extends Matiyasevich's by applying it to rounding error estimation.

In Section 2, we explain what kinds of functions we consider and clarify the concept of "computation with

---

This is a translation of the IPSJ Best Paper Award paper that appeared originally in Japanese in Transactions of IPSJ, Vol. 30, No. 7 (1989), pp. 807-815.

*Department of Administration Engineering, Faculty of Science and Technology, Keio University, Kohoku-ku, Yokohama 223, Japan.

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

$$v_1 \leftarrow \psi_1(u_{11}, u_{12})$$
$$\vdots$$
$$v_j \leftarrow \psi_j(u_{j1}, u_{j2})$$
$$\vdots$$
$$(f=)v_r \leftarrow \psi_r(u_{r1}, u_{r2})$$

$v_1, \ldots, v_r$: intermediate variables;
$\psi_1, \ldots, \psi_r$: basic operations;
u_{j1}, u_{j2}: formal parameters of ψ_j (each corresponds to an input variable, a constant, or one of the previous intermediate variables $v_1, \ldots, v_{j-1}$ as the actual parameter), $j=1, \ldots, r$.

Fig. 1 Computational process.

$$\bar{v}_1 \leftarrow \bar{\psi}_1(\bar{u}_{11}, \bar{u}_{12}) = \psi_1(\bar{u}_{11}, \bar{u}_{12}) + \delta_1$$
$$\vdots$$
$$\bar{v}_j \leftarrow \bar{\psi}_j(\bar{u}_{j1}, \bar{u}_{j2}) = \psi_j(\bar{u}_{j1}, \bar{u}_{j2}) + \delta_j$$
$$\vdots$$
$$(\bar{f}=)\bar{v}_r \leftarrow \bar{\psi}_r(\bar{u}_{r1}, \bar{u}_{r2}) = \psi_r(\bar{u}_{r1}, \bar{u}_{r2}) + \delta_r$$

$\bar{v}_1, \ldots, \bar{v}_r$: intermediate variables with finite precision;
$\bar{\psi}_1, \ldots, \bar{\psi}_r$: basic operations with finite precision;
$\bar{u}_{j1}, \bar{u}_{j2}$: formal parameters of $\bar{\psi}_j$ (each corresponds to an input variable, a constant, or one of the previous intermediate variables $\bar{v}_1, \ldots, \bar{v}_{j-1}$ as the actual parameter), $j=1, \ldots, r$;
$\delta_1, \ldots, \delta_r$: generated rounding errors.

Fig. 2 Computational process with rounding errors.

rounding errors". In Section 3, we introduce the concept of machine interval operations, describe our algorithm, and prove its optimality. In Section 4, we give the results of numerical experiments with some observations.

2. Computation with Rounding Errors

2.1 Piecewise Factorable Functions and Computational Process

We assume that the operations used in the computation of a function are either unary or binary, such as $+$, $-$, $\times$, $/$, exp, log, etc., and are continuously differentiable in their domain of definition. We call them *basic operations*. The functions we consider in this paper are the so-called *piecewise factorable functions* [9], whose values are calculated by a finite sequence of basic operations represented in the form of a procedure or a program. The procedure for computing a piecewise factorable function may contain conditional branches and iterations that depend on the values of the input variables. We call the sequence of operations, which has actually been executed in the computation of the value of the function with given input values, the *computational process*. The computational process consists of *computational steps*, each of which executes a basic operation and then stores its value in a variable called an *intermediate variable*. For a function with n variables $f(x_1, \ldots, x_n)$, its computational process is represented as shown in Fig. 1. (Hereafter, descriptions will be only for binary operations; it should be understood that, for unary operations, the descriptions regarding the second argument will be deleted.) The number of intermediate variables in a computational process is equal to the number of computational steps, r.

2.2 Floating-Point Systems and Computations with Rounding Errors

On a computer, the result of a real operation is usually approximated by the value of the corresponding floating-point operation. For a specific computer (and a compiler), the *floating-point system* means the set of floating-point numbers representable in the computer and the manner of rounding for real numbers. The so-called *machine epsilon* ε_M expresses the supremum of the relative rounding errors occurring in the floating-point system.

When we perform computation with finite precision, that is, with rounding errors, the computed values of intermediate variables as well as the computed value of the function are somewhat different from the values that would be obtained by computation with infinite precision. With finite-precision computation, it may happen that these differences in the computed values lead to different branches chosen at the conditional branches in the program and to division by a number with a different sign (which latter would imply the possibility of division by zero), and hence that the computational process realized by computation with finite precision differs in structure from that realized by computation with infinite precision. However, we shall assume in the following that the computational process remains the same in spite of the existence of rounding errors. (The validity of this assumption can be checked by means of interval analysis, as will be stated in Section 3.)

We assume that the values of input variables $x_1, \ldots, x_n$ are given. After the sequence of basic operations has been executed, giving rise to a process for computing f, the value of f at $(x_1, \ldots, x_n)$ is obtained. When the value of f is computed by using a floating-point system, rounding errors arise due to approximation. If the value of f can be computed by using interval arithmetic, as we have assumed above, a unique structure of the computational process is determined. We denote the jth computational step actually performed with rounding errors by "$\bar{v}_j \leftarrow \bar{\psi}_j(\bar{u}_{j1}, \bar{u}_{j2})$", instead of "$v_j \leftarrow \psi_j(u_{j1}, u_{j2})$" as in Fig. 1, which would have been performed in the ideal case where no rounding error occurred. $\bar{\psi}_j$ indicates a basic operation with finite precision. (Horizontal bars over letters will indicate "finite precision" in the following.) We define the *generated rounding error* δ_j associated with the jth computational step by

$$\bar{\psi}_j(\bar{u}_{j1}, \bar{u}_{j2}) = \psi_j(\bar{u}_{j1}, \bar{u}_{j2}) + \delta_j \qquad (2.2.1)$$

(see Fig. 2).

Since the actual parameters of ψ_j on the right-hand side of Eq. (2.2.1) are the computational results in finite precision, the generated rounding error δ_j is a local er-

ror "generated" in the execution of $\bar{\psi}_j$. Assuming the smoothness of basic operations, the difference between the jth computed result $\bar{v}_j$ and the exact value v_j is expressed as follows:

$$\begin{aligned}\bar{v}_j - v_j &= \bar{\psi}_j(\bar{u}_{j1}, \bar{u}_{j2}) - \psi_j(u_{j1}, u_{j2}) \\ &= \psi_j(\bar{u}_{j1}, \bar{u}_{j2}) - \psi_j(u_{j1}, u_{j2}) + \delta_j \\ &= \frac{\partial \psi_j}{\partial u_{j1}}(u_{j1} + \theta_j \cdot (\bar{u}_{j1} - u_{j1}), \\ &\qquad u_{j2} + \theta_j \cdot (\bar{u}_{j2} - u_{j2})) \cdot (\bar{u}_{j1} - u_{j1}) \\ &\quad + \frac{\partial \psi_j}{\partial u_{j2}}(u_{j1} + \theta_j \cdot (\bar{u}_{j1} - u_{j1}), \\ &\qquad u_{j2} + \theta_j \cdot (\bar{u}_{j2} - u_{j2})) \cdot (\bar{u}_{j2} - u_{j2}) + \delta_j \end{aligned} \tag{2.2.2}$$

with some θ_j between 0 and 1. $\partial\psi_j/\partial u_{j1}$ and $\partial\psi_j/\partial u_{j2}$ are the partial derivatives of the basic operation ψ_j, which we call *elementary partial derivatives* [5, 6, 7] and whose values at $(u_{j1} + \theta_j \cdot (\bar{u}_{j1} - u_{j1}), u_{j2} + \theta_j \cdot (\bar{u}_{j2} - u_{j2}))$ we denote by d_1^j and d_2^j, respectively:

$$d_i^j \equiv \frac{\partial \psi_j}{\partial u_{ji}}(u_{j1} + \theta_j \cdot (\bar{u}_{j1} - u_{j1}), u_{j2} + \theta_j \cdot (\bar{u}_{j2} - u_{j2})) \qquad (i = 1, 2). \tag{2.2.3}$$

Thus, we get

$$\bar{v}_j - v_j = d_1^j \cdot (\bar{u}_{j1} - u_{j1}) + d_2^j \cdot (\bar{u}_{j2} - u_{j2}) + \delta_j. \tag{2.2.4}$$

The *accumulated rounding error* in the computed value of the function is equal to the difference between $\bar{v}_r (= \bar{f})$ and $v_r (= f)$:

$$\bar{f} - f = \bar{v}_r - v_r = d_1^r \cdot (\bar{u}_{r1} - u_{r1}) + d_2^r \cdot (\bar{u}_{r2} - u_{r2}) + \delta_r. \tag{2.2.5}$$

We will rewrite the right-hand side of Eq. (2.2.5) with the help of a computational graph, $G = (V, E, \partial^+, \partial^-, \omega, n, d)$ [6, 10]. We consider a set P_j whose elements are directed paths from vertex v_j to vertex v_r, where v_j corresponds to the jth intermediate variable and v_r corresponds to the rth intermediate variable, that is, the function itself. Each path p in P_j is a sequence of arcs $a_1, \ldots, a_l$. The initial vertex $\partial^+ a_1$ of a_1 is v_j and the terminal vertex $\partial^- a_l$ of a_l is v_r ($\partial^- a_k = \partial^+ a_{k+1}$; $k = 1, \ldots, l-1$). We can define a sequence of pairs $\{(s_k, i_k)\}_{k=1}^{l}$ along the path $p = a_1, \ldots, a_l$ such that s_k indicates the number of the computational step corresponding to $\partial^- a_k$, i.e., $v_{s_k} = \partial^- a_k$, and i_k indicates that a_k corresponds to the formal parameter $u_{s_k i_k}$. Now denoting by Q_j the set of such sequences corresponding to directed paths in P_j, we can define

$$\begin{aligned} w_j &= \sum_{\{(s_k, i_k)\}_{k=1}^{l} \in Q_j} d_{i_l}^{s_l} \cdot d_{i_{l-1}}^{s_{l-1}} \cdot \cdots \cdot d_{i_1}^{s_1} \\ &\left(= \sum_{\{(s_k, i_k)\}_{k=1}^{l} \in Q_j} \frac{\partial \psi_r}{\partial u_{ri_l}} \cdot \frac{\partial \psi_{s_{l-1}}}{\partial u_{s_{l-1} i_{l-1}}} \cdot \cdots \cdot \frac{\partial \psi_{s_1}}{\partial u_{s_1 i_1}} \right) \end{aligned} \tag{2.2.6}$$

($j = 1, \ldots, r-1$), and $w_r = 1$. Replacing $\bar{u}_{r1} - u_{r1}$ and $\bar{u}_{r2} - u_{r2}$ in Eq. (2.2.5) with the corresponding intermediate variables and substituting the expressions (2.2.4) repeatedly, we finally get

$$\bar{f} - f = \bar{v}_r - v_r = \sum_{j=1}^{r} w_j \cdot \delta_j. \tag{2.2.7}$$

We shall discuss an algorithm for evaluating $|\bar{f} - f|$ rigorously in the following section.

If we could set $\theta_1 = 0, \ldots, \theta_r = 0$, then we would have

$$w_j = \frac{\partial f}{\partial v_j} \tag{2.2.8}$$

owing to the chain-rule for differentiation of a compound function. Thus, we have the linear approximation L_f of $\bar{v}_r - v_r$:

$$\bar{v}_r - v_r \simeq L_f \equiv \sum_{j=1}^{r} \frac{\partial f}{\partial v_j} \cdot \delta_j, \tag{2.2.9}$$

which affords a base for the *absolute bound* and *probabilistic estimate* of the rounding error in previous papers [5, 6, 7]. Here, we can practically calculate all $\partial f / \partial v_j (j = 1, \ldots, r)$ with the method of *Fast Automatic Differentiation*. Since $|\delta_j| \leq |v_j| \cdot \varepsilon_M$ (ε_M: machine epsilon) holds for almost all computers, the absolute bound A_f may be defined [5, 6, 7, 10] as

$$A_f \equiv \sum_{j=1}^{r} \left| \frac{\partial f}{\partial v_j} \right| \cdot |v_j| \cdot \varepsilon_M, \tag{2.2.10}$$

which gives a practical and good approximation to the upper bound of the absolute value of the rounding error but which is not a rigorous upper bound, because Eq. (2.2.9) is already an approximate formula.

3. An Algorithm for a Rigorous Upper Bound of the Absolute Value of the Rounding Error

We will give an algorithm for calculating an interval $S = [s^l, s^h]$ such that $\bar{f} - f \in S$. We can obtain the rigorous upper bound of the absolute value of the rounding error by $|\bar{f} - f| \leq |S|$ (where, for an interval $X \equiv [x^l, x^h]$, $|X|$ is defined to be $\max\{|x^l|, |x^h|\}$).

3.1 Interval Operations and Machine Interval Operations [1]

By a *machine interval* we shall mean an interval of real numbers $[a^l, a^h]$ such that both a^l and a^h are floating-point numbers. We sometimes refer to an interval of real numbers simply as an *interval*. An *interval operation* is an operation that takes intervals as arguments and produces an image (which is also an interval) of the intervals of arguments by the corresponding real arithmetic operation. A *machine interval operation* is an operation that takes machine intervals as arguments and produces the narrowest possible machine interval that contains the result of the corresponding interval operation with the same arguments. Of course, machine intervals and machine interval operations depend on the floating-point system.

Substituting the interval machine operations for the basic operations (and machine interval variables for the variables) in the procedure for calculating a function f, we can construct a procedure that produces machine interval $\bar{F}$ starting from the input intervals with width 0, $[\bar{x}_1, \bar{x}_1], \ldots, [\bar{x}_n, \bar{x}_n]$, where $\bar{x}_i = x_i$ for all i. $\bar{F}$ obviously contains both f and $\tilde{f}$, so the width of $\bar{F}$ is a rigorous upper bound for the absolute value of the rounding error. But it is known that the width of $\bar{F}$ may become too wide to use for practical purposes when the function f is computed in many computational steps. In the following section, we will show a method for obtaining a narrower upper bound for the absolute value of the rounding error than $\bar{F}$ after the computation of $\bar{F}$.

The history of the machine interval operations actually executed in the computation of f is the computational process in terms of machine interval operations. For a certain floating-point system and a set of input data, the "computability of $\bar{F}$ by machine interval operations" will mean that we can execute all the necessary machine interval operations without dividing by a machine interval including zero or comparing two intersecting machine intervals. (If division by a machine interval including zero or comparison between intersecting machine intervals is required, we cannot proceed any further.) Thus, if $\bar{F}$ can be computed, the value of the function will certainly be computable on the assumption stated in Section 2.2. (If $\bar{F}$ is not computable, the following arguments will be meaningless.) Evidently, if $\bar{F}$ is computable by a floating-point system represented by a machine epsilon ε_M, it is computable by any floating-point system that has a longer mantissa and a longer exponent and that therefore has a smaller machine epsilon ε'_M than ε_M, the resulting interval $\bar{F}'$ determined by the latter system being included in the interval $\bar{F}$ by the former.

3.2 Calculation of Partial Derivatives and Estimation of Rounding Errors

Having computed $\bar{F}$ by machine interval operations, we shall then proceed as follows. If we denote by $\bar{V}_j$ the machine interval corresponding to an intermediate variable v_j in the computation of the machine interval $\bar{F}$ corresponding to f, we have

$$v_j, \tilde{v}_j \in \bar{V}_j \quad (j=1, \ldots, r), \tag{3.2.1}$$

and thus

Table 1 Machine intervals corresponding to elementary partial derivatives.
$\bar{D}_i^j$ corresponds to $\partial\psi_j/\partial u_{ji}$, where $v_j=\psi_j(u_{j1}, u_{j2})|_{u_{j1}=v_{i_1}, u_{j2}=v_{i_2}}$.

ψ_j	$\bar{D}_1^j$	$\bar{D}_2^j$
$\pm$	[1, 1]	$[\pm 1, \pm 1]$
$*$	$\bar{V}_l$	$\bar{V}_k$
$/$	$[1, 1] \oslash \bar{V}_l$	$[-1, -1] \otimes \bar{V}_j \oslash \bar{V}_l$
exp	$\bar{V}_j$	—
log	$[1, 1] \oslash \bar{V}_k$	—

$$u_{ji} + \theta_j \cdot (\tilde{u}_{ji} - u_{ji}) \in \bar{U}_{ji} \quad (j=1, \ldots, r;\ i=1, 2) \tag{3.2.2}$$

where we denote by $\bar{U}_{ji}$ the machine interval corresponding to u_{ji}, and $0<\theta<1$.

First, we substitute the machine interval operations for the operations in the computation of the elementary partial derivatives $\partial\psi_j/\partial u_{ji}$. Denoting by $\bar{D}_i^j$ ($j=1, \ldots, r$; $i=1, 2$) the machine intervals calculated by those machine interval operations, we have

$$\bar{D}_i^j \ni d_i^j = \frac{\partial \psi_j}{\partial u_{ji}}(u_{j1} + \theta_j \cdot (\tilde{u}_{j1} - u_{j1}),\ u_{j2} + \theta_j \cdot (\tilde{u}_{j2} - u_{j2})) \tag{3.2.3}$$

since $0<\theta_j<1$ ($j=1, \ldots, r$). (See Table 1, where $\oplus$, $\ominus$, $\otimes$, $\oslash$, and so on indicate the machine interval operations corresponding to $+$, $-$, $\times$, $/$, and so on, respectively.)

Next, we calculate a machine interval $\bar{\Delta}_j$ containing the generated error δ_j that depends on the values of the arguments of the basic operation which computes the intermediate variable v_j. For example, since in almost all computers the absolute value of the generated error δ_j is less than $|\tilde{v}_j| \cdot \varepsilon_M$ (or more precisely, $\beta^{\lfloor \log_\beta(|\tilde{v}_j|) \rfloor} \cdot \varepsilon_M$ for an ordinary floating-point system with radix β), we may set $\bar{\Delta}_j \equiv [-\bar{\delta}_j, \bar{\delta}_j]$ with $\bar{\delta}_j \equiv |\tilde{v}_j| \cdot \varepsilon_M$.[1] Thus, from Eq. (2.2.5), we have

$$\tilde{f} - f \in (\bar{D}'_1 \otimes (\tilde{u}_{r1} - u_{r1}) \oplus \bar{D}'_2 \otimes (\tilde{u}_{r2} - u_{r2})) \oplus \bar{\Delta}_r. \tag{3.2.4}$$

Expanding $\tilde{u}_{r1} - u_{r1}$, $\tilde{u}_{r2} - u_{r2}$ in a similar manner repeatedly, we obtain the interval formula

$$\tilde{f} - f \in ((\cdots(\bar{W}_r \otimes \bar{\Delta}_r) \oplus \cdots) \oplus \bar{W}_2 \otimes \bar{\Delta}_2) \oplus \bar{W}_1 \otimes \bar{\Delta}_1 \tag{3.2.5}$$

corresponding to Eq. (2.2.7). $\bar{W}_j$ ($j=1, \ldots, r$) are the machine intervals that correspond to w_j in (2.2.6). They are computed by substituting the machine interval operations for the operations in the algorithm for Fast Automatic Differentiation [5, 6, 7, 10] as follows:

1) Initialization: —
   $\bar{W}_1, \ldots, \bar{W}_{r-1} := [0, 0]$;
   $\bar{W}_r := [1, 1]$
2) Computation: —
   for $j := r$ downto 1 do
   $a :=$ index of the intermediate variable of the first actual parameter of ψ_j;
   $b :=$ index of the intermediate variable of the second actual parameter of ψ_j;
   $\bar{W}_a := \bar{W}_a \oplus \bar{W}_j \otimes \bar{D}_1^j$;
   $\bar{W}_b := \bar{W}_b \oplus \bar{W}_j \otimes \bar{D}_2^j$.

(When ψ_j has only one argument or when an actual parameter of ψ_j is not an intermediate variable, either a or b is not defined and the corresponding operation is

[1] Considering possible underflows, we may set $\bar{\delta}_j \equiv \max\{|\tilde{v}_j| \cdot \varepsilon_M, pmin, -nmax\}$ where *pmin* and *nmax* are the minimum positive floating-point number and the maximum negative floating-point number, respectively, representable in the floating-point system.

omitted. Note that, if we can execute the machine interval operations for f to obtain $\bar{F}$, we can compute all $\bar{W}_j$ without dividing by a machine interval containing zero and without comparing intersecting intervals. Furthermore, we may compute the machine interval products $W_s \otimes \bar{D}_t^i$, and so on, when they become necessary, instead of computing $\bar{D}_t^i$ in advance [5, 6].) It is important to note that all $\bar{W}_1, \ldots, \bar{W}_r$ here can be computed in a time proportional to r.

Thus, if we define A_F by

$$A_F \equiv \left| \bigoplus_{j=1}^{r} \bar{W}_j \otimes \bar{\Delta}_j \right|,$$

then $|\bar{f}-f| \leq A_F$ holds rigorously, where $\oplus$ indicates summation by machine interval additions.

3.3 Algorithm for Computing A_F and its Computational Complexity

The algorithm for computing A_F can be summarized as follows:

(1) The machine interval operations are substituted for the real basic operations appearing in the procedure for computing the value of f, and the machine interval variables for the real variables;

(2) The procedure constructed in (1) is executed with $\bar{X}_1=[\underline{x}_1, \bar{x}_1], \ldots, \bar{X}_n=[\underline{x}_n, \bar{x}_n]$ as inputs, and the computational process for computing $\bar{V}_1, \ldots, \bar{V}_r(=\bar{F})$ is simultaneously determined;

(3) $\bar{W}_1, \ldots, \bar{W}_r$ are computed by means of the algorithm of Fast Automatic Differentiation, where the intervals for elementary partial derivatives may be computed either beforehand or at the same time as the computation of $\bar{W}_j$'s;

(4) $\bar{\Delta}_j \equiv [-\bar{\delta}_j, \bar{\delta}_j]$, where $\bar{\delta}_j = |\bar{V}_j| \varepsilon_M$ $(j=1, \ldots, r)$ (or $|\bar{v}_j| \varepsilon_M$ when $\bar{v}_j$ is calculated together with $\bar{V}_j$);

(5) $S \equiv \bigoplus_{j=1}^{r} \bar{W}_j \otimes \bar{\Delta}_j$ ($\oplus$: machine interval summation);

(6) $A_F \equiv |S|$.

The time complexity of each machine interval operation is proportional to that of the corresponding ordinary real basic operation. Therefore, (1) and (2) above are performed in a time proportional to that needed for calculating the function alone. The computation of (3) is also performed in a time proportional to that for calculating the function by means of Fast Automatic Differentiation. Hence, the time for computing A_F is proportional to that for calculating the function f alone. By a similar argument, the space required to compute A_F is also proportional to the time (not to the space) for calculating f. (There have been some proposals to reduce the size of space, for example, by multi-level decomposition of the computational graph [13].)

3.4 Optimality Property

Let us denote by $R=|\bar{f}-f|$ the absolute value of the rounding error in $\bar{f}$. Then, we have

$$R = \left| \sum_{j=1}^{r} w_j \cdot \delta_j \right| \tag{3.4.1}$$

(see Eq. (2.2.7)). On the assumption in Section 1, it may happen that all the signs of $w_1 \cdot \delta_1, \ldots, w_r \cdot \delta_r$ coincide with one another. In this case, we have

$$R = \sum_{j=1}^{r} |w_j| \cdot |\delta_j|. \tag{3.4.2}$$

If we denote by $\bar{\delta}_j$ ($|\delta_j| \leq \bar{\delta}_j$) the estimate that is a floating-point number and the tight upper bound for the absolute value of δ_j, we must anticipate the worst case, in which R is as large as

$$\bar{R} = \sum_{j=1}^{r} |w_j| \cdot \bar{\delta}_j. \tag{3.4.3}$$

Since A_F gives the width of a machine interval that is guaranteed to be no smaller than $\bar{R}$, we have $\bar{R} \leq A_F$.

Furthermore, on the assumption that

$$\partial f / \partial v_j \neq 0 \quad (j=1, \ldots, r), \tag{3.4.4}$$

we can prove that A_F is sharp enough, that is, it is not too large in comparison with $\bar{R}$. More specifically, we shall prove that, as the machine epsilon ε_M approaches zero, that is, as the length of the mantissa part of the floating-point system becomes longer with a sufficient number of bits for the exponent part, the ratio $A_F/\bar{R}$ tends to 1.

Theorem. Let

$$A_F \equiv \left| \bigoplus_{j=1}^{r} \bar{W}_j \otimes \bar{\Delta}_j \right|,$$

$$\bar{R} \equiv \sum_{j=1}^{r} |w_j| \cdot \bar{\delta}_j \quad \text{and} \quad \bar{\Delta}_j \equiv [-\bar{\delta}_j, \bar{\delta}_j]. \tag{3.4.5}$$

Then, for an arbitrary $\varepsilon > 0$, there exists $\eta > 0$ such that, if the computation with machine interval operations is performed with $\varepsilon_M < \eta$, we have

$$1 \leq \frac{A_F}{\bar{R}} \leq 1+\varepsilon. \tag{3.4.6}$$

□

Proof. The following (i) holds because of the theorems[1] for machine interval operations:

(i) For any $\varepsilon' > 0$ there exists an η_1 (>0) such that, in a floating-point system with any machine epsilon ε_M less than η_1, w_j and $\bar{W}_j$ which we defined in Sections 2.2 and 3.2, satisfy the relations:

$$0 \leq |\bar{W}_j| - |w_j| \leq |\bar{W}_j - w_j| \leq \varepsilon' \quad (j=1, \ldots, r). \tag{3.4.7}$$

Moreover, because of assumption (3.4.4), the continuous differentiability of f (in some domain), and the computability of $\bar{F}$, the following (ii) holds:

(ii) For $m \equiv \frac{1}{2} \min_j \left| \frac{\partial f}{\partial v_j} \right|$ (>0), there exists η_2 such that

[1] Theorems 4 and 5 in Chapter 4 of Alefeld and Herzberger [1].

w_j computed in a floating-point system with any machine epsilon ε_M less than η_2 satisfies

$$|w_j| > m \quad (j=1, \ldots, r).$$

From (i) and (ii), we have

(iii) For $\eta_3 \equiv \min\{\eta_1, \eta_2\}$,

$$1 \le \frac{|\bar{W}_j|}{|w_j|} \le 1 + \frac{\varepsilon'}{m}, \tag{3.4.8}$$

where w_j and $\bar{W}_j$ are computed in a floating-point system with machine epsilon ε_M less than η_3.

It is trivial to show that $A_F/\bar{R} \ge 1$, from the definition of A_F. Furthermore, it may be seen that, even if we take account of the rounding errors generated during the calculation of the inner product with the machine interval operations in Section 3.3 (5), we have the inequality

$$A_F \le \left(\sum_{j=1}^{r} |\bar{W}_j| \cdot \bar{\delta}_j \right) \cdot (1+\varepsilon_M)^r. \tag{3.4.9}$$

In fact, the absolute value of the product $\bar{W}_j \otimes \bar{\delta}_j$ is bounded by $|\bar{W}_j| \cdot \bar{\delta}_j \cdot (1+\varepsilon_M)$; for each addition in the inner product, the upper bound of the intermediate result is multiplied by $1+\varepsilon_M$; and there are $r-1$ additions in the inner product.

For any positive $\varepsilon(<1)$, let $\varepsilon' = \varepsilon \cdot m/2$ and determine η_3 as in (iii). Then, if we compute the algorithm (Section 3.3) in a floating-point system with machine epsilon ε_M less than $\eta_4 \equiv \min\{\eta_3, \varepsilon/4r\}$, we have

$$\begin{aligned} \frac{A_F}{\bar{R}} &\le \frac{\sum_{j=1}^{r} |\bar{W}_j| \cdot \bar{\delta}_j \cdot (1+\varepsilon_M)^r}{\sum_{j=1}^{r} |w_j| \cdot \bar{\delta}_j} \\ &\le \left(1 + \frac{\varepsilon'}{m}\right) \cdot (1 + e^{r\varepsilon_M} \cdot r \cdot \varepsilon_M) \\ &\le 1 + \frac{\varepsilon'}{m} + 1.5 \cdot e^{1/4} \cdot \frac{1}{4} \cdot \varepsilon \\ &\le 1 + \varepsilon, \end{aligned} \tag{3.4.10}$$

where we use the inequalities $(1+x)^r \le e^{rx} \le 1 + e^{rx} \cdot r \cdot x$ $(x \ge 0,\ r \ge 0)$. □

4. Numerical Experiments

4.1 Simulation of Machine Interval Operations

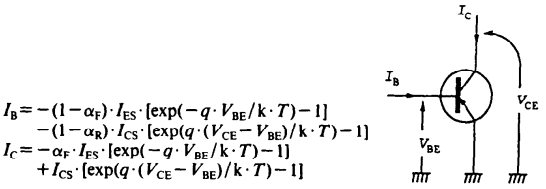
We have neither special hardware nor software for direct execution of interval machine operations such as PASCAL-SC [2], so we simulated machine interval operations on a VAX8600 machine with Kyoto Common Lisp (KCl) on the ULTRIX operating system, as follows. The radix of the native floating-point system is binary, the mantissa of a number consists of 56 bits and the rounding is toward the nearest. The machine epsilon is equal to 2^{-56}. On the machine, we used a program to implement binary floating-point systems that can have a mantissa of any length less than 56 bits. They are represented in terms of their machine epsilon, ε_M. The interval machine operations in a floating-point system represented by ε_M were performed as follows. For each machine interval operation,

(1) we compute a machine interval $C \equiv [c^l, c^h]$ in the floating-point system represented by ε_M according to the definition of the interval operation;

(2) then, we calculate the machine interval $\bar{C} \equiv [\bar{c}^l, \bar{c}^h]$ that includes the perturbation of C by ε_M, such that

$$\begin{aligned} \bar{c}^l = \min\{&c^l \cdot (1-\varepsilon_M), c^l \cdot (1+\varepsilon_M), \\ &c^h \cdot (1-\varepsilon_M), c^h \cdot (1+\varepsilon_M)\}, \\ \bar{c}^h = \max\{&c^l \cdot (1-\varepsilon_M), c^l \cdot (1+\varepsilon_M), \\ &c^h \cdot (1-\varepsilon_M), c^h \cdot (1+\varepsilon_M)\}. \end{aligned}$$

For simplicity, we neglected the effect of underflow. Note that there may be narrower machine intervals than $\bar{C}$, which contain the result of the machine interval operation.



$$I_B = -(1-\alpha_F) \cdot I_{ES} \cdot [\exp(-q \cdot V_{BE}/k \cdot T) - 1] - (1-\alpha_R) \cdot I_{CS} \cdot [\exp(q \cdot (V_{CE} - V_{BE})/k \cdot T) - 1]$$

$$I_C = -\alpha_F \cdot I_{ES} \cdot [\exp(-q \cdot V_{BE}/k \cdot T) - 1] + I_{CS} \cdot [\exp(q \cdot (V_{CE} - V_{BE})/k \cdot T) - 1]$$

Fig. 3 Ebers-Moll model for a pnp-transistor.

I_B, I_C: base current and collector current
I_{ES}, I_{CS}: saturation currents for the emitter-base junction and the collector-base junction
α_F, α_R: current transfer ratios
V_{BE}, V_{CE}: voltages with the emitter as the datum node
T: temperature
q: electric charge of an electron
k: Boltzmann constant

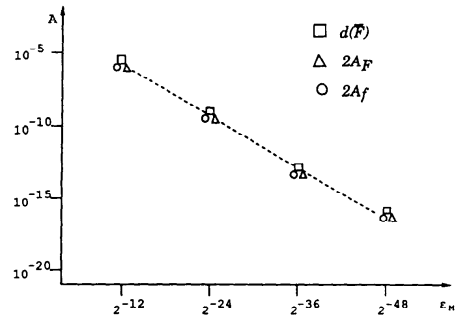


Fig. 4 Comparison of the widths of guaranteed intervals containing the exact value for the function that is the base current in the Ebers-Moll model.
$I_B \simeq -1.04 \times 10^{-4}$ A, where $V_{BE} = -0.4$ V, $V_{CE} = -1.0$ V, $I_{ES} = 1.0 \times 10^{-9}$ A, $I_{CS} = 2.0 \times 10^{-9}$ A, $\alpha_F = 0.98$, $\alpha_R = 0.5$, $T = 300$ K, $q = 1.602 \times 10^{-19}$ C, k $= 1.38066 \times 10^{-23}$ J/K

4.2 Example 1: The Ebers-Moll Model of a Transistor

We regarded the expression of the *base current* I_B in the Ebers-Moll model of a pnp-type transistor (Fig. 3) as the definition of a function. The width of the

Table 2 Guaranteed intervals with $\bar{F}$ and A_F for a 10-dimensional linear system.

ε_M	computed value and significant digits (underlined) estimated by means of A_f	guaranteed interval with $\bar{F}$	guaranteed interval with A_F
2^{-12}	0.7438964843	—	—
2^{-24}	0.7542193532	[−3344.4582519, 3346.1162109]	[0.4637820125, 1.0446566939]
2^{-36}	0.7542197853	[0.0575954438, 1.4508441332]	[0.7542197555, 0.7542198151]
2^{-48}	0.7542197855	[0.7540463030, 0.7543932679]	[0.754219785475757, 0.754219785484078]

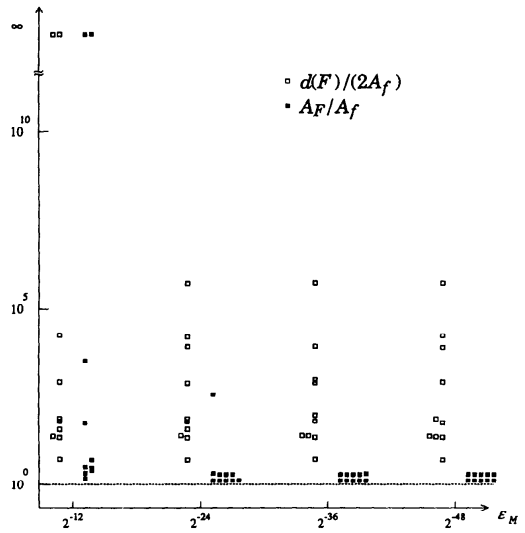


Fig. 5 Comparison of the widths of guaranteed intervals for 5-dimensional linear systems.
Points whose ordinates are positioned at the symbol "∞" indicate that the calculations corresponding to those points were interrupted by the occurrence of a division by an interval containing zero.

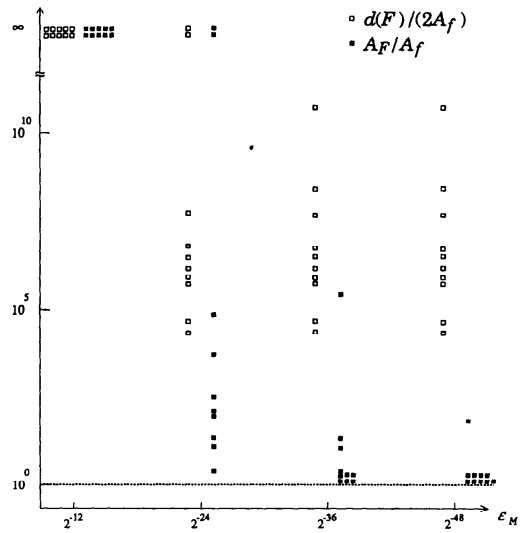


Fig. 6 Comparison of the widths of guaranteed intervals for 10-dimensional linear systems.
Points whose ordinates are positioned at the symbol "∞" indicate that the calculations corresponding to those points were interrupted by the occurrence of a division by an interval containing zero.

guaranteed interval given by the estimate A_F proposed in this paper is $2 \cdot A_F$ and that of the result $\bar{F}$ with machine interval operations is $d(\bar{F})$. The linearly approximated width of the guaranteed interval is $2 \cdot A_f$, where A_f is the absolute bound (Eq. (2.2.10)). They are compared for four different machine epsilons (ε_M in Section 4.1)—2^{-12}, 2^{-24}, 2^{-36}, and 2^{-48}—as shown in Fig. 4. There it is observed that $2 \cdot A_f$, $2 \cdot A_F$ and $d(\bar{F})$ are almost equal for this specific case of a small-scale function, and that there is apparently no purpose in computing A_F after computing $\bar{F}$.

4.3 Example 2: Solutions of Linear Systems by Means of LU Decomposition

A program for solving a linear system $Ax=b$ for x with matrix A and vector b as input data by means of LU decomposition was regarded as a program for computing x as a set of functions with variables A and b. Specifically, we regarded the first component x_1 of x as the value of a function $f(A, b)$ with $n(n+1)$ variables, where n is the dimension of x. We prepared ten pairs of a 5×5 matrix and a 5-dimensional vector $((A_1, b_1), \ldots, (A_{10}, b_{10}))$ whose components were independently sampled from the uniform distribution on $[-1, 1]$. We calculated A_f, A_F and $d(\bar{F})$ to estimate the rounding errors occurring in the values of $x_{1i}=f(A_i, b_i)$ $(i=1, \ldots, 10)$ for the four machine epsilons mentioned in Section 4.1.

There were cases in which we could not compute $\bar{F}$ (and therefore could not computer A_F) because of large rounding errors for large machine epsilons. Therefore, we chose the absolute bound of linear approximation, A_f, as the basis for comparison. Figure 5 shows A_F/A_f and $d(\bar{F})/(2 \cdot A_f)$. In that figure, those points whose ordinates are positioned at the symbol "∞" indicate that the calculations corresponding to those points were interrupted by the occurrence of a division by an interval containing zero. We also carried out similar experiments for 10-dimensional matrices and vectors (Fig. 6). In Table 2, a part of the results of the latter ex-

periments (the 10-dimensional case) is shown numerically, and it can be seen that the number of significant digits for the intervals guaranteed by A_F is markedly greater than the number for the intervals guaranteed by $\bar{F}$.

It may be observed that, in most cases, the widths of machine intervals obtained by naïve machine interval operations are between 10^1 and 10^5 times larger for 5-dimensional linear systems, and between 10^4 and 10^8 times larger for 10-dimensional linear systems, respectively, than those obtained by the method proposed in this paper. In summary, it can be said that our estimate A_F gives not only a rigorous upper bound but also a sharp upper bound of the absolute value of the rounding error. In usual situations (where ε_M is less than 2^{-24} for 5-dimensional linear systems, and less than 2^{-48} for 10-dimensional linear systems), it is also observed that A_f is a good practical approximation of the upper bound of the absolute value of the rounding error.

Although sometimes hardware and software [2] that compute inner products without rounding errors are available, we considered here a situation in which each of the multiplications and the additions in computing the inner product of vectors generates a rounding error individually.

5. Conclusion

We proposed a practicable algorithm for estimating a rigorous and sharp upper bound of the absolute value of the rounding error occurring in the computed value of a function.

Through our numerical experiments we have observed that the estimate of rounding error based on linear approximation is usually good enough, but it is important, at least theoretically, to establish a technology that gives a rigorous and sharp estimate.

References

1. ALEFELD, G. and HERZBERGER, J. *Introduction to Interval Computations*. Academic Press, New York (1983).
2. BOHLENDER, G., ULLRICH, C., GUDENBERG, J. W. and RALL, L. B. *Pascal-SC—A Computer Language for Scientific Computation*. Academic Press, Orlando (1987).
3. GRIEWANK, A. On Automatic Differentiation. M. Iri and K. Tanabe (eds.): *Mathematical Programming—Recent Developments and Applications*, Kluwer Academic Publishers (1989), 83−107.
4. HANSEN, E. R. A Generalized Interval Arithmetic. K. Nickel (ed.): *Interval mathematics*, Lecture Notes in Computer Science 29, Springer-Verlag, Berlin (1975), 7-18.
5. IRI, M. Simultaneous Computation of Functions, Partial Derivatives and Estimates of Rounding Errors—Complexity and Practicality. *Japan Journal of Applied Mathematics*, **1**, 2 (1984), 223-252.
6. IRI, M. and KUBOTA, K. Methods of Fast Automatic Differentiation and Applications. *Research Memorandum RMI 87-02*, Department of Mathematical Engineering and Information Physics, University of Tokyo (1987).
7. IRI, M., TSUCHIYA, T. and HOSHI, M. Automatic Computation of Partial Derivatives and Rounding Error Estimates with Applications to Large-scale Systems of Nonlinear Equations. *Journal of Computational and Applied Mathematics*, **24**, 3 (1988), 365-392.
8. IRI, M. and KUBOTA, K. Norms, Rounding Errors, Partial Derivatives and Fast Automatic Differentiation. *Transactions of the Institute of Electronics, Information and Communication Engineers (Japan)*, E**74**, 3 (1991), 463-471.
9. KEDEM, G. Automatic Differentiation of Computer Programs. *ACM Trans. Math. Softw.*, **6**, 2 (1980), 150-165.
10. KUBOTA, K. and IRI, M. Formulation and Analysis of Computational Complexity of Fast Automatic Differentiation (in Japanese). *Trans. IPS Japan*, **29**, 6 (1988), 551-560.
11. LINNAINMAA, S. Taylor Expansion of the Accumulated Rounding Error. *BIT*, **16** (1976), 146-160.
12. MATIYASEVICH, Yu. V. Veshchestvennye Chisla i ÈVM. *Kibernetika i Vychislitel'naya Tekhnika*, Vypusk 2 (1986), 104-133.
13. VOLIN, Yu. M. and OSTROVSKII, G. M. Automatic Computation of Derivatives with the Use of the Multilevel Differentiating Technique—I. Algorithmic Basis. *Computers and Mathematics with Applications*, **11**, 11 (1985), 1099-1114.
14. RALL, L. B. Improved Bounds for Ranges of Functions. K. Nickel (ed.): *Interval Mathematics 1985*, Lecture Notes in Computer Science 212, Springer-Verlag, Berlin (1985), 143-155.