Automatic computation of partial derivatives and rounding error estimates with applications to large-scale systems of nonlinear equations *

Masao IRI and Takashi TSUCHIYA **
Department of Mathematical Engineering and Instrumentation Physics, Faculty of Engineering, University of Tokyo, Bunkyo-ku, Tokyo 113, Japan

Mamoru HOSHI
Department of Information Processing Engineering, Faculty of Engineering, Chiba University, Chiba-shi, Chiba 260, Japan

Received 4 February 1988
Revised 8 March 1988

Abstract: Recently, a new approach has been proposed to efficiently compute the accurate values of partial derivatives of a function or functions, and simultaneously to estimate the rounding errors in the computed function values. In this paper the use of the method in the solution of nonlinear equations is investigated.

The method makes use of the computational graph and, when applied to the evaluation of a function, traverses it from the top (= the function node) down to the bottom (= the input variable nodes). A remarkable analogy is observed between the partial derivatives and the shortest paths on the computational graph. The top-down traversing on the computational graph has the following advantages over the existing algorithms using the bottom-up traversing: (1) The gradient of a function can be computed within the same complexity as that of the evaluation of the function alone (the complexity being independent of the number of input variables); (2) A fairly sharp estimate of the rounding error in the function evaluation is obtained, on the basis of which a computationally meaningful norm may be introduced in the space of residuals to afford a convergence criterion for an iterative method of solving the system of nonlinear equations. As an example, a system of nonlinear equations with 108 variables for a distillation tower of a chemical plant is numerically analyzed in detail. It is shown that by the use of the proposed method we could satisfactorily resolve two main problems encountered in computing a numerical solution of the system of nonlinear equations, i.e., how to compute the accurate Jacobian matrix and when we should stop the iteration.

Keywords: Fast automatic differentiation algorithm, rounding error estimates, systems of nonlinear equations, optimization problems, computational graph, a norm and metrics induced by the rounding errors.

* This paper is a completely improved and updated English version of the original Japanese paper by the same authors, selected and published in the Silver-Jubilee special issue (Vol. 26, No. 11 (1985), pp. 1411 - 1420) of the Journal of Information Processing Society of Japan.
** Presently, the Institute of Statistical Mathematics, Minato-ku, Tokyo 106, Japan.
1. Introduction

In numerical computation, many fundamental concepts are borrowed from pure mathematics. However, it is often difficult to numerically realize those pure-mathematical concepts exactly so that some sort of approximation is indispensable. Thus concepts in numerical computations are often more or less different from the corresponding ones in pure mathematics. Existence of rounding errors in the numerical process would be the primary source of the difference, from which we cannot get away. In this paper, we propose a practically implementable method for coping with some of fundamental difficulties encountered in computing a numerical solution of nonlinear equations.

The first problem we take up in this paper is an efficient method for computing the Jacobian matrix of a system of nonlinear functions. In many practical algorithms for solving a system of nonlinear equations such as the Newton method, it is necessary to compute the Jacobian matrix repeatedly. The methods widely used so far are either writing out explicitly the programs for computing the derivatives or resorting to the so-called numerical differentiation using the program for the functions only. However, the former method is too laborious to manually perform even for moderate-size real-world industrial or social problems as we have seen by experience. To obtain the programs for the derivatives without great effort, we may make use of a programming language, such as REDUCE, MACSYMA, etc., for symbolic differentiation. But, in this case, we would have a tremendous amount of formulae for a large system, usually proportional to the square of the size of the original system or more, and it would take much time to compute them. Furthermore, when the program for the function contains "IF" statements or "DO" loops, the differentiation in the form of formulae would not be easy to perform automatically.

The so-called "numerical differentiation" is certainly "the" alternative method to overcome such difficulties of the former method. However, it is well known that numerical differentiation has the drawback of being highly susceptible to rounding errors. Although there is a theory telling us how to choose the optimum value of $h$ in an approximation like $[f(x + h) - f(x)]/h$ for $f'(x)$, it is still an annoying problem in practical situations how to choose the value $h$ because the optimum value is different for different functions and for different variables. Even if we could know the optimum $h$, we lose a substantial part of the significant digits. Furthermore, the function must be evaluated many times, ordinarily at least as many times as the number of variables plus one.

The second problem considered in this paper, which might be less apparent but actually not less important, is what kind of stopping criterion for iteration to adopt. We have traditionally used a certain norm in the space of residuals, i.e., $\| f \| = (\sum f_i^2)^{1/2}$, max $|f_i|$, $\sum |f_i|$ or the like, to observe the convergence of the iterative process for solving the system of equations $f_i(x_1, \ldots, x_n) = 0 \ (i = 1, \ldots, n)$. But, the mathematical and physical meaning of the norm of this type seems to require thorough reflection. For example, when $f_1 = 0$ is an equation for the equilibrium of force, $f_2 = 0$ represents the law of conservation of mass and $f_3 = 0$ represents the heat balance in a physical system, how could we compare or add the quantities of different physical dimensions to have a physically significant quantity? From another more mathematical standpoint, it is the fact that all the norms on $\mathbb{R}^n$ are equivalent, and there are infinite possibilities of choosing "weights" to get a "weighted" norm of the form $\| f \|_w = \max w_i |f_i|$, $\sum w_i |f_i|$, etc. All of them are mathematically equivalent with respect to the convergence to
“zero”. However, numerically, i.e., in the presence of noise of rounding errors, norms with different weights would often behave quite differently. Then, which norm among infinitely many possibilities is the most appropriate from a numerical standpoint? This may be interpreted as a problem of scaling the equations. In this connection, it may be plausible to regard the problem of solving the system of equations \( f_i(x_1, \ldots, x_n) = 0 \) \((i = 1, \ldots, n)\) numerically as that of finding an \( x \) such that \( |f_i(x)| \leq \Delta f_i(x) \) for all \( i \), where \( \Delta f_i(x) \) is a sufficiently sharp upper bound for the absolute value of the rounding error incurred in the computation of the value of \( f_i \) for that value of \( x \). Then, \( w_i = 1 / \Delta f_i(x) \) is evidently a proper choice in the “weighted” norm (note that \( w_i \) may depend on \( x \)). The type of the norm, \( \max \cdot \cdot \cdot \cdot \cdot \cdot \cdot \) or \( (\Sigma \cdot \cdot \cdot \cdot \cdot \cdot \cdot)^{1/2} \), would be less important.

With these weights in the definition of the norm, we may stop the iteration whenever the norm is reduced to the order of magnitude of 1, \( n \), \( \sqrt{n} \), respectively. Furthermore, this norm defined in the space of residuals will induce a kind of metrics in the space of solution \( x \), which represents the inherent uncertainty in the numerical solution.

Thus, in order to define a numerically meaningful norm, it is crucial to find a practical method by which we may obtain sharp estimates for the rounding errors incurred in the function evaluation. Then interval analysis may be reminded of as such a method (see, e.g., [2]). However, it does not usually give estimates sharp enough because it cannot easily take into account the phenomenon of cancellation of rounding errors. On the other hand, the established theory of rounding errors which reflects faithfully the process of generation and propagation of rounding errors [3,16] has been regarded as a mere theory, but is not practical, because it requires us to compute partial derivatives of the functions with respect to all the intermediate variables appearing in their computation, which would be a formidable task for complicated functions without efficient means for partial differentiation.

Recently, a new algorithm has been proposed with which we can resolve these two difficulties in numerical computation [5,8]. According to the algorithm which is based on the analogy between partial derivatives and shortest paths on an acyclic graph representing the computational process, we can compute the accurate values of all the partial derivatives of each of the functions with respect to all the intermediate variables in time proportional to that required for evaluating the original functions alone, the coefficient of proportionality being independent of the number of input variables. This means that we can get at the same time the accurate values of the entries of the Jacobian matrix and all the informations that are necessary for defining a reasonable norm such as the above.

In this paper, extensive experimental study will be carried out to demonstrate the usefulness of the above-mentioned algorithm for the solution of the system of nonlinear equations, and it will be shown that the difficulties in numerically solving nonlinear equations, which we pointed out in the above, can be overcome. Specifically, it will be shown by way of computational experiments that the accurate Jacobian matrix can be computed efficiently and that the norm defined with weights associated with rounding error estimates works nicely.

2. Algorithmic background

To begin with we shall illustrate the algorithm with a simple example for the function \( f(x, y) = (x \cdot y + b) \cdot x + c \). As shown in (1) ~ (4) of Fig. 1, the process for computing the value of the function \( f \) for given values of the input variables \( x, y \) can be expressed as a sequence of
computations of intermediate variables $v_1, v_2, v_3$ and the function $f$. (Throughout this paper, an intermediate variable will also mean an input variable or an output variable (i.e., a function), and will be denoted by $v_i$. When we need to distinguish explicitly an input or an output variable from the other intermediate variables, we shall denote it by $x_i$ or $f_i$, respectively.) Thus the procedure of computing a function is generally represented by a computational scheme, i.e., a sequence of basic computational steps, as

$$v_i = \phi_i(u_{i1}, \ldots, u_{im}), \quad i = 1, \ldots, n_v, \quad (2.1)$$

where $\phi_i$ is one of the basic operations (+, −, ×, /, sin, cos, exp, log, etc) a priori defined, and each of $u_{ik}$ ($k = 1, \ldots, m_i$) is (i) an input variable, (ii) a constant, or (iii) an intermediate variable already computed.

A computational scheme can be equivalently represented by a computational graph, where an intermediate variable $v_i$ in the computational scheme corresponds to a vertex in the computational graph and the vertex corresponding to the variable $v_i$ is connected by an arc from each of the vertices corresponding to the arguments $u_{i1}, \ldots, u_{im}$, of $\phi_i$ in (2.1). The graph thus constructed is called the computational graph for the function, and, obviously, it is an acyclic graph. It is also easy to see that, given a computational graph, the corresponding computational scheme is determined up to an unessential reordering. That is, the computational graph is another, more intrinsic expression of the computational procedure for functions. Figure 2 is the computational graph corresponding to the computational scheme of (1)-(4) in Fig. 1.

To compute the partial derivatives, we first attach the elementary partial derivative $\frac{\partial v_i}{\partial u_{ij}}$ to every arc connecting $u_{ij}$ ($j = 1, \ldots, m_i$) and $v_i$ on the computational graph. Then, in terms of the elementary partial derivatives, the partial derivative of the function $f$ with respect to an input variable $x_i$ can be expressed as

$$\frac{\partial f}{\partial x_i} = \sum \frac{\partial f}{\partial v_{i1}} \cdot \frac{\partial v_{i1}}{\partial x_i} \cdot \frac{\partial v_{i2}}{\partial x_i} \cdot \ldots \cdot \frac{\partial v_{il}}{\partial x_i}, \quad (2.2)$$

by the chain rule for the derivative of a composite function, where the sum of the products on the right-hand side is taken over all the directed paths $(x_i, v_{il}, \ldots, v_{i1}, f)$ from $x_i$ to $f$, i.e., the
partial derivative of \( f \) with respect to \( x_i \) is obtained by first computing the product of the elementary partial derivatives along each path from the vertex corresponding to \( x_i \) to the vertex corresponding to \( f \) and then taking the sum of those products along all such paths.

There is a noticeable analogy between formula (2.2) and the well-known formula for the shortest distance defined on the graph having the same topological structure as the computational graph:

\[
d(f, x_i) = \min \{ d(f, u_{i1}) + \ldots + d(u_m, x_i) \},
\]

where \( d(f, x_i) \) is the distance along the shortest directed path from \( x_i \) to \( f \) in the graph with the "arc-distance function \( d \)" defined on the arc set taking the place of the elementary partial derivatives. Thus, it is seen that formulae (2.2) and (2.3) differ from each other only in that the product and the sum in (2.2) are replaced, respectively, by the sum and the minimum in (2.3). Then, due to the well-known analogy between algebraic systems \((\times, +)\) and \((+, \min)\), we can interpret the computation of partial derivatives as a variation of the shortest path problem for an acyclic graph [1]. Then it is readily seen that "Given the elementary partial derivatives, we can compute the partial derivatives of a function with respect to all the intermediate variables including the input variables with a complexity at most constant (which is independent of the number of input variables) times as large as that of evaluating the function alone." When all of the basic operations are unary or binary, the following relation holds between the number \( V \) of the vertices and the number \( A \) of the arcs on the computational graph:

\[
A \leq 2V.
\]

Fig. 2. Computational graph for the function "\( f = (y \cdot x + b) \cdot x + c \)".
Thus, the number of basic operations needed to compute all the elementary partial derivatives is no more than twice as many as that necessary to compute the function alone. Then it is easy to see that we can compute all the partial derivatives of a function with a complexity a constant times as large as that for the computation of the original function.

The algorithm for computing the partial derivatives of a function after the evaluation of the function is executed, is as follows:

\begin{algorithm}
\begin{itemize}
  \item \textbf{Step 0.} Compute the elementary partial derivatives.
  \item \textbf{Step 1.} (Compute the partial derivatives of the function with respect to all the intermediate variables.)
    \begin{itemize}
      \item Initialization:
        \begin{itemize}
          \item [for all $v_i$] $\frac{\partial f}{\partial v_i} := 0$;
          \item $\frac{\partial f}{\partial f} := 1$.
        \end{itemize}
      \end{itemize}
    \end{itemize}
  \item Computation of the partial derivatives:
    \begin{itemize}
      \item (In the reverse order of the execution of the basic computational steps in evaluating the function.)
      \item \textbf{for each basic computational step} $v_i = \phi_i(u_{i1}, \ldots, u_{im_i})$ do
        \begin{itemize}
          \item \textbf{for $k := 1$ to $m_i$ do}
            \begin{itemize}
              \item $\frac{\partial f}{\partial u_{ik}} := \frac{\partial f}{\partial u_{ik}} + \frac{\partial f}{\partial v_i} \cdot \frac{\partial v_i}{\partial u_{ik}}$.
            \end{itemize}
        \end{itemize}
    \end{itemize}
\end{itemize}
\end{algorithm}

The steps (5) - (10) in Fig. 1 are the computational procedure of partial derivatives of the function of our example according to this algorithm. In the case where several functions are simultaneously computed in a computational scheme, we may repeat Step 1 for each of those functions. In the rest of this paper we refer to this algorithm for derivatives as the \textit{fast differentiation algorithm}.

Methods for computing derivatives based on the program for computing a function or functions have been devised and proposed by many authors, since the first attempt was made by Wengert [15]. For example, in the book [13] by L.B. Rall, a historical survey is given together with the description of the system developed by himself. It is interesting to see that most of them adopt the bottom-up traversing (in our terminology) of the computational graph starting from the input variables to the output variables, whereas a few adopt the top-down traversing like ours [5,10,11,12,14]. These two ways of traversing make a remarkable contrast in time complexity when computing the gradient of a single function. That is, by the bottom-up traversing, the graph needs to be traversed as many times as there are input variables, so that the total complexity would be proportional to the number of input variables. On the other hand, by the top-down traversing, the graph is traversed only once, so that the complexity is independent of the number of input variables. This kind of complexity argument seems to be first stated in [4] for rational functions in order to derive good lower bounds for the complexity of some algebraic computational problems.

The space complexity also depends on the way of traversing. In the top-down algorithm, computations of the functions and their derivatives are carried out in the opposite directions on
the computational graph, one starting from the input variables and the other from the output variables. Therefore, we have to preserve all the values of the intermediate variables and/or those of the elementary partial derivatives obtained in the stage of computation of the functions, in addition to the topology of the computational graph. Thus the space complexity of the top-down algorithm is the same in order as the number of steps in the function evaluation. On the other hand, in the bottom-up algorithm, if computation of the derivatives is carried out for each input variables separately and the function evaluation is repeated each time, then we do not have to preserve all the information of the computational graph, so that the space complexity is of the same order as that of the function evaluation.

There is another more important advantage of the top-down algorithm over the bottom-up, i.e., with the former algorithm, we can obtain fairly sharp estimates for the bounds of rounding error contained in the computed values of the functions as a by-product of the algorithm as will be shown in the following section. With the bottom-up, we may apply a method similar to interval analysis to obtain rounding-error estimates, which, however, cannot take account of the possible cancellation of errors along different paths, and hence cannot but be less sharp.

3. Rounding error estimates

In the following, we shall denote the computed values of \( v, f \), etc. by \( \tilde{v}, \tilde{f} \), etc., respectively, and the operation, which is actually carried out in finite precision corresponding to a basic operation \( \phi \), by \( \tilde{\phi} \). In general, for each computational step, the discrepancy \( \Delta u_i \) (to be called the accumulated rounding error) of the computed value \( \tilde{u}_i \) from the value \( u_i \) which would have been obtained in the “ideal” computation is written as follows:

\[
\Delta u_i = \tilde{u}_i - u_i
\]

Assuming that \( \Delta u_{i_k} - \tilde{u}_{i_k} - u_{i_k} \) etc. are small enough, we can approximate (3.1) by (3.2) using elementary partial derivatives:

\[
\Delta v_i = \sum_k \frac{\partial v_i}{\partial u_{i_k}} \Delta u_{i_k} + \delta v_i, \quad (3.2)
\]

\[
\delta v_i = \tilde{\phi}_i(\tilde{u}_{i_1}, \ldots, \tilde{u}_{i_m}) - \phi_i(u_{i_1}, \ldots, u_{i_m}), \quad (3.3)
\]

where the first term and the second term on the right-hand side of (3.2) are sometimes called the propagated error and the generated error, respectively. By applying (3.2) to the whole computational procedure of a function, we get the following representation (3.4) for the accumulated rounding error in the computed value of the function:

\[
\Delta f = \sum_i \frac{\partial f}{\partial v_i} \delta v_i, \quad (3.4)
\]

where the sum is taken over all the intermediate variables except the input variables.
Since the algorithm described in the preceding section enables us to compute all $\partial f / \partial u_i$'s efficiently, a practical estimate of $\Delta f$ can be obtained without difficulty as follows. For that purpose, some more assumption is necessary. Specifically, the latest computational environment with a digital computer seems to allow us to assume (and hence we shall do so) that

(i) computation is done with floating-point arithmetic in which a real number $u_i$ is represented as

$$v_i = \text{sign}(v_i) \times M \times \beta^e,$$

where $e$ is an integer “exponent” and $M$ is a $L$-digit “mantissa” with base $\beta$ (usually $\beta = 2, 10$ or 16) which is normalized in such a way that $\beta^{-1} \leq M < 1$, and

(ii) the result of every basic operation is rounded to the “nearest” number representable in the form of (3.5) (so that the result is accurate up to one unit of the least significant digit of $M$ in (3.5)).

The precise bound for $\Delta v_i$ will depend on the meaning of “nearest” in (ii). When “nearest” is taken in the one-sided neighborhood, we have

$$|\Delta v_i| \leq \epsilon_i = \beta^{1-L} \cdot m(v_i) \leq \beta^{1-L} \cdot |v_i|,$$

whereas, when the “nearest” is taken in the two-sided neighborhood, we have

$$|\Delta v_i| \leq \epsilon_i = \beta^{1-L} \cdot \frac{m(v_i)}{2} \leq \beta^{1-L} \cdot \frac{|v_i|}{2},$$

where $m(v)$ is the smallest nonnegative number in the normalized floating-point representation which has the same exponent as $v$ has. Obviously,

$$m(v) \leq |v| < \beta \cdot m(v)$$

holds. The quantity $\beta^{1-L}$ or $\beta^{1-L}/2$ in (3.6) or (3.7) is usually called the unit of rounding or the machine epsilon, and is denoted by $\epsilon$. (The $\epsilon_i$ in (3.6) and (3.7) can be written in the form:

$$\epsilon_i = \epsilon \cdot m(v_i) \leq \epsilon \cdot |v_i|$$

in terms of the machine epsilon $\epsilon$.)

Here let us note that it is a well-accepted assumption in the theory of rounding errors that

(iii) $\Delta v_i$ is an instance of a random variable subject to the uniform distribution over the interval $[0, \epsilon_i]$ (or $(-\epsilon_i, 0]$) or $[-\epsilon_i, \epsilon_i]$ corresponding to (3.6) and (3.7), respectively. Furthermore, the random variables for different intermediate variables are independent of one another.

Based on these assumptions and equations (3.4), (3.6) and (3.7), we can readily derive the following estimates for the rounding error $\Delta f$ of $f$.

(1) Absolute bound: From (3.4), (3.6) and (3.7), we can obtain the following estimates for the rounding error of $f$:

$$|\Delta f| \leq \sum_i \left| \frac{\partial f}{\partial v_i} \Delta v_i \right|$$

$$\leq \epsilon \sum_i \left| \frac{\partial f}{\partial v_i} \right| \cdot m(v_i) = A_1[f, x] \cdot \epsilon$$

$$\leq \epsilon \sum_i \left| \frac{\partial f}{\partial v_i} \right| \cdot |v_i| = A_2[f, x] \cdot \epsilon,$$
where

$$A_1[f, x] = \sum_i \left| \frac{\partial f}{\partial v_i} \right| \cdot m(v_i), \quad (3.12)$$

$$A_2[f, x] = \sum_i \left| \frac{\partial f}{\partial v_i} \right| \cdot |v_i|. \quad (3.13)$$

We shall call $A_1$ and $A_2$ the normalized absolute bounds for the rounding error of $f$, where “normalized” means the normalization with respect to the machine epsilon $\epsilon$. The product of the normalized absolute bounds and the machine epsilon will be called simply the absolute bounds.

(2) Probabilistic bound: From (3.4) we obtain another more practical type of estimates for the error bound. In fact, regarding $\Delta f$ and $\delta v_i$ themselves as random variables, we have at once

$$E[\Delta f] = \sum_i \frac{\partial f}{\partial v_i} E[\delta v_i], \quad (3.14)$$

$$V[\Delta f] = \sum_i \left| \frac{\partial f}{\partial v_i} \right|^2 V[\delta v_i]. \quad (3.15)$$

$$E[(\Delta f)^2] = \sum_i \left| \frac{\partial f}{\partial v_i} \right|^2 E[\delta v_i^2] + \sum_{i \neq j} \frac{\partial f}{\partial v_i} \frac{\partial f}{\partial v_j} E[\delta v_i] E[\delta v_j]$$

$$= E[\Delta f]^2 + V[\Delta f], \quad (3.16)$$

where $E$ and $V$ denote the expectation and the variance, respectively, of a random variable. In the following, we shall derive two probabilistic estimates $P_1 \cdot \epsilon$, the “exact” estimate for $E[(\Delta f)^2]$, and $P_2 \cdot \epsilon$, a reasonable approximation to $P_1 \cdot \epsilon$.

In the case where $\delta v_i$ obeys the uniform distribution over $[-\epsilon, \epsilon]$ ($\epsilon_i = m(v_i) \cdot \epsilon$), we have $E[\delta v_i] = 0$ (and consequently $E[\Delta f] = 0$) and $V[\delta v_i] = \epsilon^2/3$ so that we have

$$E[(\Delta f)^2] = V[\Delta f] = \frac{1}{3} \epsilon^2 \sum_i \left| \frac{\partial f}{\partial v_i} \right|^2 \cdot m(v_i)^2 = P_1[f, x]^2 \cdot \epsilon^2 \quad (3.17)$$

$$\leq \frac{1}{3} \epsilon^2 \sum_i \left| \frac{\partial f}{\partial v_i} \right|^2 \cdot |v_i|^2 = P_2[f, x]^2 \cdot \epsilon^2, \quad (3.18)$$

where

$$P_1[f, x] = \left[ \frac{1}{3} \sum_i \left| \frac{\partial f}{\partial v_i} \right|^2 \cdot m(v_i) \right]^{1/2}, \quad (3.19)$$

$$P_2[f, x] = \left[ \frac{1}{3} \sum_i \left| \frac{\partial f}{\partial v_i} \right|^2 \cdot |v_i|^2 \right]^{1/2}. \quad (3.20)$$

In the case where $\delta v_i$ obeys the uniform distribution over $[0, \epsilon]$ (or $(-\epsilon, 0]$) ($\epsilon_i = m(v_i) \cdot \epsilon$), we have $E[\delta v_i] = +\text{sign}(v_i)\epsilon$ and $V[\delta v_i] = \epsilon^2/12$.

$$E[(\Delta f)^2] = \frac{\epsilon^2}{3} \sum_i \left| \frac{\partial f}{\partial v_i} \right|^2 \cdot m(v_i)^2 + \frac{\epsilon^2}{4} \sum_{i \neq j} \text{sign}(v_i) \text{sign}(v_j) \frac{\partial f}{\partial v_i} \frac{\partial f}{\partial v_j} m(v_i) m(v_j)$$

$$= P_1[f, x]^2 \cdot \epsilon^2. \quad (3.21)$$
where

\[ P_1[f, x] = \left[ \frac{1}{3} \sum_i \left( \frac{\partial f}{\partial v_i} \right)^2 m(v_i)^2 + \frac{1}{4} \sum_{i \neq j} \text{sign}(v_i) \text{sign}(v_j) \frac{\partial f}{\partial v_i} \frac{\partial f}{\partial v_j} m(v_i) m(v_j) \right]^{1/2}. \]

(3.22)

However, it is not very easy to obtain a practically meaningful bound for (3.22) using \(|v_i| \cdot \varepsilon\), because the signs before the summands of the second term in the square root are not easy to determine in general. Nevertheless, it may not be so unrealistic to expect that the second term will not be very large compared with the first due to the probable cancellation in the sum of many quantities of different signs. Hence, neglecting the second term in the square root and substituting \(|v_i| \cdot \varepsilon\) for \(m(v_i) \cdot \varepsilon\), we have the same expression as (3.18), i.e., we adopt the expressions (3.20) as the “definition” of \(P_2\) in the case of chopping, too.

We shall call \(P_1\) and \(P_2\) the normalized probabilistic bounds for the rounding error of \(f\), and the products \(P_1 \cdot \varepsilon\) and \(P_2 \cdot \varepsilon\) the probabilistic bounds. When we want as sharp an estimate as possible, the bound \(A_1\) or \(P_1\) is preferable, whereas, when we think more of a simple computation than a sharp estimate, \(A_2\) or \(P_2\) may work well. \(A_2\) or \(P_2\) gives us an overestimate over \(A_1\) or \(P_1\) at worst by a factor of 2 (in the case of binary floating-point representation) or 16 (in the case of hexadecimal representation).

When \(\delta v_i\)’s obey the uniform distribution over \([-\varepsilon, \varepsilon]\) independently of one another, we can in principle get estimates for “covariances” between rounding errors of different functions in case there are many functions. From the assumptions (i)–(iii) and (3.4), we have the following expression for the covariance between \(\Delta f_i\) and \(\Delta f_j\) when all the basic operations are followed by the rounding in the two-sided neighborhood:

\[ \text{cov}[\Delta f_i, \Delta f_j] = \sum_k \frac{\partial f_i}{\partial v_k} \frac{\partial f_j}{\partial v_k} V[\delta v_k] = \frac{1}{2} \varepsilon^2 \sum_k \frac{\partial f_i}{\partial v_k} \frac{\partial f_j}{\partial v_k} m(v_k)^2. \]  

(3.23)

4. Preliminary experiment on the method with a small example—Ebers–Moll model for a pnp-transistor

In this section we shall show some results of a preliminary experiment with a small example to investigate the efficiency of the fast differentiation algorithm and the accuracy of rounding error estimates. The example we take up is the so-called “Ebers–Moll model” of a pnp-type transistor, consisting of mathematical functions representing the relation between the voltages and the currents of the transistor. The functions are as shown in Fig. 3, where the transistor is assumed to operate with the emitter grounded. There are two functions \(I_B\) and \(I_c\) and nine input variables \(V_{BE}, V_{CE}, I_{ES}, I_{CS}, \alpha_F, \alpha_R, T, q\) and \(k\). Throughout the experiment we set the values of input variables approximately as follows:

\[ I_{ES} = 1.0 \times 10^{-9} \text{A}, \quad I_{CS} = 2.0 \times 10^{-9} \text{A}, \quad \alpha_F = 0.98, \quad \alpha_R = 0.5, \quad V_{BE} = -0.4 \text{V}, \quad V_{CE} = -1.0 \text{V}, \quad T = 300.0 \text{K}, \quad q = 1.602 \times 10^{-13} \text{C}, \quad k = 1.38053 \times 10^{-23} \text{J/K}. \]  

(4.1)
Here we note that all the experiments in this paper were done on HITAC M-280H (14MIPS; VOS3 FORTRAN77) and DEC VAX-11/780 (1MIPS; UNIX FORTRAN77) of the Computer Center of the University of Tokyo.
Table 1
Comparison of the computation times of the functions and their derivatives by IM (Intermediate-variable method) and OD (Ordinary method)

(a) Compiler optimization level = OPT(0)

<table>
<thead>
<tr>
<th></th>
<th>CPU times (μs)</th>
<th>Ratio of CPU times (1)</th>
<th>Ratio of CPU times (2)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>IM</td>
<td>OD</td>
<td>IM</td>
</tr>
<tr>
<td>F</td>
<td>14</td>
<td>25</td>
<td>F</td>
</tr>
<tr>
<td>F&amp;J</td>
<td>35</td>
<td>194</td>
<td>F&amp;J</td>
</tr>
</tbody>
</table>

(b) Compiler optimization level = OPT(1)

<table>
<thead>
<tr>
<th></th>
<th>CPU times (μs)</th>
<th>Ratio of CPU times (1)</th>
<th>Ratio of CPU times (2)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>IM</td>
<td>OD</td>
<td>IM</td>
</tr>
<tr>
<td>F</td>
<td>13</td>
<td>20</td>
<td>F</td>
</tr>
<tr>
<td>F&amp;J</td>
<td>26</td>
<td>140</td>
<td>F&amp;J</td>
</tr>
</tbody>
</table>

(c) Compiler optimization level = OPT(2)

<table>
<thead>
<tr>
<th></th>
<th>CPU times (μs)</th>
<th>Ratio of CPU times (1)</th>
<th>Ratio of CPU times (2)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>IM</td>
<td>OD</td>
<td>IM</td>
</tr>
<tr>
<td>F</td>
<td>10</td>
<td>16</td>
<td>F</td>
</tr>
<tr>
<td>F&amp;J</td>
<td>24</td>
<td>100</td>
<td>F&amp;J</td>
</tr>
</tbody>
</table>

4.1. Experiment on the efficiency of computation

(a) Procedure: Two programs for computing the functions were prepared: One describes the functions with the original expressions in Fig. 3 as they are, and the other optimizes the computational procedure as is shown in Fig. 4 introducing 17 additional intermediate variables. (The details of this example are shown in [5].) The execution times for computing the values of the functions using these two programs were measured and compared.

Then, the values of the functions and their derivatives (i.e., the elements of the Jacobian matrix) were computed by two methods—one computing each derivative independently of the other with the expression derived from the function by manually differentiating the expression for the function (to be abbreviated as “OD (Ordinary method)”), and the other performing the computation of the derivatives by the fast differentiation algorithm in Section 2 (to be abbreviated as “IM (Intermediate-variable method)”) and the execution times by the two methods were measured and compared. All the timing data here were measured on M-280H.

(b) Result: As shown in Table 1. It is worth noting that the ratios of the execution time by OD to that by IM were 1.5–1.8 for the functions only (see the row “F (Functions)”), 4.5–5.5 for the functions and the Jacobian matrix (see the row “F & J (Functions and Jacobian matrix)”).

4.2. Experiment on rounding errors

We investigated the validity of the three assumptions (i)–(iii) in Section 3 by comparing the observed statistical values with the theoretical. The computational experiment was done on
M-280H (hexadecimal, chopping, $\epsilon = 16^{-5}$ for single precision) and VAX-11/780 (binary, rounding, $\epsilon = 2^{-24}$ for single precision).

(a) Procedure: Rounding errors in the two functions computed with single precision were assumed to be approximately equal to their differences from the values computed with double precision. We computed 1000 samples of rounding errors by slightly perturbing the normal value of $V_{\text{BE}}$; specifically we set $V_{\text{BE}} = - 0.4 + n \times \Delta V_{\text{BE}}$ ($n = 1, \ldots, 1000$), where $\Delta V_{\text{BE}}$ was chosen so that the lower digits of the function values may be slightly changed, leaving the values of the other input variables untouched. The average and the standard deviation of the samples thus obtained were compared with their theoretical estimates based on the mathematical formulae (3.14) and (3.15) for $E[\Delta f]$ and $V[\Delta f]$, where we adopted a more elaborated assumption for errors generated in the computation of intermediate variables, i.e., we took one-point distribu-

<table>
<thead>
<tr>
<th>Computer</th>
<th>HITAC M-280H (hexadecimal, chopping)</th>
<th>DEC VAX-11/780 (binary, rounding)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$I_B \doteq -1.0 \times 10^{-4}$ A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rounding error in $I_B$</td>
<td>Average Observed value</td>
<td>$-2.01 \times 10^{-10}$ A</td>
</tr>
<tr>
<td></td>
<td>Theoretical estimate (3.14)</td>
<td>$-1.73 \times 10^{-10}$ A</td>
</tr>
<tr>
<td>Standard deviation</td>
<td>Average Observed value</td>
<td>$3.83 \times 10^{-10}$ A</td>
</tr>
<tr>
<td></td>
<td>Theoretical estimate (3.15)</td>
<td>$3.80 \times 10^{-10}$ A</td>
</tr>
<tr>
<td>Observed maximum rounding error</td>
<td>$0.93 \times 10^{-9}$ A</td>
<td>$1.80 \times 10^{-10}$ A</td>
</tr>
<tr>
<td>Absolute bound for rounding error (3.11)</td>
<td>$5.24 \times 10^{-9}$ A</td>
<td>$3.28 \times 10^{-10}$ A</td>
</tr>
</tbody>
</table>

| $I_C \doteq -5.2 \times 10^{-3}$ A |                                       |                                   |
| Rounding error in $I_C$ | Average Observed value | $-0.65 \times 10^{-8}$ A | $0.26 \times 10^{-8}$ A |
|                          | Theoretical estimate (3.14)       | $-0.55 \times 10^{-8}$ A | $0.24 \times 10^{-8}$ A |
| Standard deviation      | Average Observed value | $1.88 \times 10^{-8}$ A | $0.28 \times 10^{-8}$ A |
|                          | Theoretical estimate (3.15)       | $1.87 \times 10^{-8}$ A | $0.27 \times 10^{-8}$ A |
| Observed maximum rounding error | $0.42 \times 10^{-7}$ A | $0.92 \times 10^{-8}$ A |
| Absolute bound for rounding error (3.11) | $2.52 \times 10^{-7}$ A | $1.57 \times 10^{-8}$ A |
tions for the intermediate variables which are not affected by the above-mentioned perturbation. The practical absolute bound $A_2 \cdot \epsilon$ were also computed for the sake of comparison with the maximum of the observed rounding errors.

(b) Result: As shown in Table 2, the theoretical values are seen to be in good agreement with the experimental results. Furthermore, the advantage of the binary computation of real numbers over the hexadecimal is apparent here, too. We also investigated the validity of the estimate of covariance (3.23) by computing the observed covariance between $\Delta l_a$ and $\Delta l_c$ and the estimate, to find a good agreement between the two. Specifically, we got $7.20 \times 10^{-18}$ as the observed covariance, whereas the theoretical estimate was $7.06 \times 10^{-18}$, in single precision with VAX-11/780.

5. Application to a large-scale system—mathematical model of a distillation tower of a chemical plant

In this section we shall deal with a computational problem of computing the solution of a larger system of nonlinear equations by the Newton method. The system is a mathematical model for computing an equilibrium state of a distillation tower, which is one of the typical real-world problems which have been treated by a commercial chemical-process simulator called DPS (Dynamic Process Simulation) developed by a software company JUSE [9]. Specifically, there are 108 equations, i.e., 108 equilibrium conditions for 108 variables characterizing the state of the system. The number of intermediate variables appearing in the computational scheme is 2743, not including the input variables, the functions and constants.

5.1. Experiment on the efficiency of computation

To compare the efficiency of the new method with that of the now widely-used numerical differentiation, we measured the computation times of the Jacobian matrix of the system in single precision by the two methods on M-280H (14MIPS). The size of the Jacobian matrix is $108 \times 108$. The computation of the elements of the Jacobian matrix by the fast differentiation algorithm was done as described in Section 2, whereas the approximations by numerical differentiation were computed by means of the formula

$$\frac{\partial f_i}{\partial x_j} \approx \frac{f_i(x_1, \ldots, x_j + h, \ldots, x_{108}) - f_i(x_1, \ldots, x_j, \ldots, x_{108})}{h},$$

where $h$ was fixed for all $i, j$ throughout. Hence, for numerical differentiation, we had to repeat the computation of the functions $108 + 1 = 109$ times. The computation times on M-280H are shown in Table 3. It is seen that the fast differentiation algorithm is about 6 times as fast as the numerical differentiation. The ratio was not significantly dependent on the optimization level of the FORTRAN compiler.

In order to analyze the computation times in more detail we measured the following:

(1) In the fast differentiation algorithm:

(a) the time for the computation of the functions and the elementary partial derivatives,
Table 3
Comparison of the computation times of the Jacobian matrix by ① the fast differentiation algorithm and by ② the numerical differentiation

<table>
<thead>
<tr>
<th>Method</th>
<th>Compiler optimization level</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>OPT(1)</td>
</tr>
<tr>
<td>① Fast differentiation</td>
<td>200 ms</td>
</tr>
<tr>
<td>algorithm</td>
<td></td>
</tr>
<tr>
<td>② Numerical differentiation</td>
<td>1180 ms</td>
</tr>
<tr>
<td>②/①</td>
<td>5.9</td>
</tr>
</tbody>
</table>

(b) the time for the computation of the Jacobian matrix, in which the gradient of each function is computed by traversing the computational graph from the top downwards:

(2) In the numerical differentiation:

(c) the time for the computation of the values of the functions $f_i(x_1, \ldots, x_{108})$ ($i = 1, \ldots, 108$) and that for the values $f_i(x_1, \ldots, x_j + h, \ldots, x_{108})$ ($i = 1, \ldots, 108; j = 1, \ldots, 108$);

(d) the time for the $108 \times 108$ subtractions and divisions.

The observed computation times (a), (b), (c) and (d) are shown in Table 4, where it is seen that most of the computation time was spent at (b) in the fast differentiation algorithm and at (c) in the numerical differentiation. In (b) of the fast differentiation algorithm, the computational graph is traversed 108 (the number of functions) times, and, in (c) of the numerical differentiation, the computational graph is traversed 109 times, nearly equally often. However, in the numerical differentiation, the “whole” computational graph must be traversed each time, whereas, in the fast differentiation algorithm, it is usually sufficient to traverse only that “part” of the computational graph which lies between a function and the input variables. In the specific example we investigated, the fast differentiation algorithm scanned only 332 intermediate variables at the maximum for a function, 1 at the minimum, and 151 on the average, out of the 2743 intermediate variables existing in the whole computational graph. Taking account of the fact that the computation of the gradient of a function requires operations approximately twice in number as the number of intermediate variables related to that function, the timing data in Table 4

Detailed analysis of the computation times of the Jacobian matrix (on M-280H, compiler optimization level = OPT(2))

<table>
<thead>
<tr>
<th>Method</th>
<th>(a)</th>
<th>(b)</th>
<th>(c)</th>
<th>(d)</th>
</tr>
</thead>
<tbody>
<tr>
<td>① Fast differentiation</td>
<td>17 ms</td>
<td>167 ms</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>algorithm</td>
<td>(9%)</td>
<td>(91%)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>② Numerical differentiation</td>
<td>-</td>
<td>-</td>
<td>1137 ms</td>
<td>8 ms</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(99%)</td>
<td>(1%)</td>
</tr>
</tbody>
</table>

Table 4
Detailed analysis of the computation times of the Jacobian matrix (on M-280H, compiler optimization level = OPT(2))

<table>
<thead>
<tr>
<th>Method</th>
<th>(a)</th>
<th>(b)</th>
<th>(c)</th>
<th>(d)</th>
</tr>
</thead>
<tbody>
<tr>
<td>① Fast differentiation</td>
<td>17 ms</td>
<td>167 ms</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>algorithm</td>
<td>(9%)</td>
<td>(91%)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>② Numerical differentiation</td>
<td>-</td>
<td>-</td>
<td>1137 ms</td>
<td>8 ms</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(99%)</td>
<td>(1%)</td>
</tr>
</tbody>
</table>
Table 3 is in good agreement with those statistics on "sparsity". It may be expected that the larger the problem is the greater will the advantage of the fast differentiation algorithm be from the viewpoint of timing.

5.2. Experiment on rounding errors

In this subsection we shall compare the theoretical estimates of rounding errors of the 108 functions with the computational results. The estimates based on $A_2 \cdot \epsilon$ and $P_2 \cdot \epsilon$ ((3.11) and (3.18)) are taken for the theoretical estimates because they are easier to compute in practical situations although $A_1 \cdot \epsilon$ and $P_1 \cdot \epsilon$ are superior from the viewpoint of the sharpness of estimates. The computations were carried out on VAX-11/780 (binary, rounding, $\epsilon = 2^{-24}$ for single precision) and M-280H (hexadecimal, chopping, $\epsilon = 16^{-5}$ for single precision).

(a) Procedure: To see the behavior of the rounding errors for the 108 functions at a given point $x = (x_1, \ldots, x_{108})$, we measured rounding errors of the 108 functions at 100 points in the vicinity of $x$. They were computed as the differences between the values computed in single precision and those in double precision. We shall refer to the maximum absolute value of the observed rounding errors of $f_i$ as the observed maximum rounding error of that function. On the other hand, we also computed the theoretical rounding error estimates $A_1 \cdot \epsilon$ and $P_1 \cdot \epsilon$ at $x$ by (3.11) and (3.18) in order to compare them with the observed maximum rounding errors for the 108 functions. We chose, as $x$, the initial approximate solution (AS1) (see Section 7) for the Newton method in §7. We made the same experiment also on the other type of estimates $A_1 \cdot \epsilon$ and $P_1 \cdot \epsilon$ based on (3.10), (3.17) and (3.21), the result of which will be shown in Appendix.

(b) Result: We first show the relation between the observed rounding errors at the point $x$ and the estimates $A_2[f_i, x] \cdot \epsilon$ and $P_2[f_i, x] \cdot \epsilon$ for the 108 functions (Fig. 5). Figure 6 shows the relation between the observed maximum rounding errors in the vicinity of $x$ and the estimates. Histograms of the ratios of the observed maximum rounding errors to the theoretical estimates for the 108 functions are shown in Fig. 7.

(c) Discussion: As shown in Fig. 5, the range of rounding errors incurred in a computation of the 108 functions is extremely wide and the ratio of the maximum to the minimum among the 108 functions is as great as $10^{11} - 10^{12}$ in this problem. It is interesting to see that such a fatally ill-scaled situation takes place in the equations representing a innocent-looking common real-world existing system. On the other hand, when compared with the width of the range of distribution of the rounding errors in the equations, the discrepancy between the rounding error estimate and the observed rounding error for each function is not large. This suggests that the estimates will work well. The sharpness of the estimates is shown more clearly in Fig. 6, where the observed maximum rounding errors (the maximum among 100 samples for each function) are compared with the estimates. There most of the points are located quite close to the line with slope 45° where the observed value is equal to the estimate.

We shall observe the results in more detail.

1. Absolute bound: From Figs. 7(a) and (b), it is observed that in the case of VAX-11/780, the absolute bounds are 3.7 times as large as the observed maximum rounding errors on the average (9 times at the maximum), and in the case of M-280H, they are 9.4 times as large as the observed maximum rounding errors on the average (25 times at the maximum). Since overestimate by a factor of 2 or 16 is considered to be inevitable under the effect of the rougher estimate of generated rounding error, these estimates may be regarded to give sufficiently sharp bounds.
Fig. 5. Comparison of the observed rounding errors $|\Delta f_1|$ with the theoretical estimates $A_2[f, x] \cdot \epsilon$ and $P_2[f, x] \cdot \epsilon$ for the 108 functions. (Solid line: observed value = estimate; broken line: observed value = estimate/10; chained line: observed value = estimate/100.) (a) VAX-11/780, absolute bound (3.11). (b) M-280H, absolute bound (3.11). (c) VAX-11/780, probabilistic bound (3.18). (d) M-280H, probabilistic bound (3.18).

(2) Probabilistic bound: It is observed that, in the case of VAX-11/780, the ratios of the observed maximum rounding errors to the estimates range between 1.0–2.7 (Fig. 7(c)). This means that the estimate $P_2 \cdot \epsilon$ is an almost exact estimate of the standard deviation of the rounding errors. On the other hand, in the case of M-280H, the ratios are distributed between
Fig. 6. Comparison of the observed maximum rounding errors \( \max |\Delta f_i| \) (among 100 samples) with the theoretical estimates \( A_2[f_i, x] \cdot \epsilon \) and \( P_2[f_i, x] \cdot \epsilon \) for the 108 functions. (Dotted line: observed value = estimate \( \times 10 \); solid line: observed value = estimate; broken line: observed value = estimate \( /10 \); chained line: observed value = estimate \( /100 \).) (a) VAX-11/780, absolute bound (3.11). (b) M-280H, absolute bound (3.11). (c) VAX-11/780, probabilistic bound (3.18). (d) M-280H, probabilistic bound (3.18).

0.3–1.4 (Fig. 7(d)). This shows that we can get estimates which are “sharp” in the sense that they are at most several times as large as the observed rounding errors, even if we work with the hexadecimal floating-point computation.
Fig. 7. Distributions of the ratios of the observed maximum rounding errors $\max |\Delta f_i|$ (among 100 samples) to the theoretical estimates $A_2[f_i, x] \cdot \epsilon$ and $P_2[f_i, x] \cdot \epsilon$ for the 108 functions. (a) VAX-11/780, absolute bound (3.11). (b) M-280H, absolute bound (3.11). (c) VAX-11/780, probabilistic bound (3.18). (d) M-280H, probabilistic bound (3.18).

From the facts observed above, we may conclude that both of the estimates proposed here are sharp enough for practical use.

6. A norm based on the estimates of rounding errors

In Introduction, we remarked that there are some serious problems related to the stopping rule or the norm used in the numerical solution of a system of nonlinear equations

$$f_i(x_1, \ldots, x_n) = 0, \quad i = 1, \ldots, n.$$  \hspace{1cm} (6.1)

The ordinary stopping rule for an iterative method, for example, the rule “Stop the iteration when a norm such as

$$\| f \| = \max_i |f_i|$$  \hspace{1cm} (6.2)

becomes less than some small constant $\delta$ given a priori by the user,” is not satisfactory nor reliable at all in the real computational situations. In this section we shall illustrate the importance of this problem concerning the stopping rule clearly through the example of the chemical plant and show how we may resolve the problem by using the good estimates of rounding errors obtained by the fast differentiation algorithm.

In the example of the distillation tower with 108 functions in Section 5, the observed rounding errors are distributed over an extremely wide range from $10^{-9}$ to $10^{-7}$ in the neighborhood of the solution. In such a case, a norm like (6.2) cannot be expected, except for chance, to become less than the magnitude of $10^2$, which is the largest (absolute) value of the
rounding error of the function among the 108 functions. The first drawback of the stopping rule by (6.2) comes from the fact that any choice of $\delta$ smaller than $10^2$ may fail to stop the iterative process. Usually we do not take $\delta$ to be so large, and hence are likely to fail to stop the iteration. This shows that, without any information about the magnitude of rounding errors, it is difficult to determine $\delta$ appropriately in advance so that the iterative process may be safely stopped.

The second drawback, which seems to be more serious, is that, for such a highly ill-scaled problem, the stopping rule by (6.2) can give no meaningful solution. We must note that some of the functions can be reduced to as small as $10^{-7}$ in value. Even if we succeeded to give an appropriate value for $\delta$ ($\approx 10^2$), it most likely happens that the iteration is stopped before the values of the functions having smaller rounding errors are reduced to be small enough. In the worst case, we may get a false solution in which all functions have the values around $10^2$, while some of them can (or should) be reduced furthermore to the magnitude of $10^{-7}$. Then, what is the meaning of stating “We successfully solved the system of equations with respect to such and such norm criterion!”? This will be the case even when we adopt a norm of another kind.

This example also tells us that the discussion based on physical dimensions in the introduction is really important. In this example, the function $f_{85}$ represents the heat balance and $f_{47}$ represents the total mass balance on a plate of the distillation tower. They have a rounding error of magnitude $10^1$ and $10^{-6}$, respectively, at the equilibrium state. The value of $f_{85}$ is measured in unit [kcal/(kg-mol \cdot h)] and that of $f_{47}$ is measured in unit [kg-mol/h]. In case we measure $f_{85}$ in [kcal/(g-mol \cdot s)] and $f_{47}$ in [g-mol/s], we may get a different solution corresponding to the new norm (or scaling) employed in the stopping rule, which is never a desirable situation.

A conceptually satisfactory way to avoid this undesirable situation is to define the problem of solving the system of equations (6.1) as the problem of finding $x$ such that

$$\| f \|_{\overline{\Delta}^{-1}} = \max_i | f_i / \overline{\Delta f}_i |,$$

with the reciprocals of the estimates of rounding errors as the weights. Since this norm takes the value of order 1 if and only if the numerical solution satisfies the condition (6.3), we can safely stop the iteration whenever the value of the norm reduces nearly equal to 1. Here, it should be noted that this new norm is physically a dimensionless quantity. Now that we can make use of sharp bounds for the rounding errors by the use of the fast differentiation algorithm, we are in the position of practically dealing with this new norm.

In the next section, we shall experimentally examine how it works effectively in solving the system of nonlinear equations.

7. Comparison of the ordinary norm and the new norm in the Newton method

We take up again the system of nonlinear equations with 108 variables in Section 5, and numerically solve it by the Newton method with the accurate (non-approximate) Jacobian
matrix. The computational experiment is designed in such a way that we can compare the iterative process based on the new norm with that based on the ordinary norm.

(a) Procedure: In the following, we shall denote the max-norm with the weight vector $w = (w_1, \ldots, w_n)^t$ by $\| f \|_w$ (i.e., $\| f \|_w = \max_i |w_i f_i|$), and the Jacobian matrix by $J$. We use the Newton method with deceleration to solve the system of equations as follows.

\textbf{(Procedure of the Newton method with deceleration)}

\textbf{Step 0.} Choose an initial approximate solution $x^{(0)}$ and compute $f(x^{(0)})$.

\textbf{Step 1.} (Computation of $Ax^{(i)}$ and the weight vector $w(x^{(i)})$.)

\[ \Delta x^{(i)} := -J^{-1}f(x^{(i)}) ;\]

compute $w(x^{(i)})$.

\textbf{Step 2.} (Deceleration)

\[ \mu := 1 ;\]

repeat

\[ y := x^{(i)} + \mu \Delta x^{(i)} ;\]

\[ \mu := \frac{1}{2} \mu \]

until $\| f(y) \|_{w(x^{(i)})} \leq (1 - \mu) \| f(x^{(i)}) \|_{w(x^{(i)})}$;

\[ x^{(i+1)} := y .\]

\textbf{Step 3.} (Check the convergence of the solution.)

\textbf{if} the condition for convergence is satisfied

\textbf{then} stop;

\textbf{else} $i := i + 1$; goto \textbf{Step 1}.

To show the advantage of the new norm, we compared four iteration processes by using the following two norms $\langle \text{OD} \rangle$ and $\langle \text{NN} \rangle$, and two initial values $\langle \text{AS1} \rangle$ and $\langle \text{AS2} \rangle$.

\textbf{(Norm)}

The ordinary norm $\langle \text{OD} \rangle$: We can realize the most widely used norm like (6.1) by taking $w$ to be $(1, \ldots, 1)^t$. We shall denote this norm simply by $\| f \|$.

The normalized norm with respect to the estimates of rounding error $\langle \text{NN} \rangle$: On the basis of the discussion in the preceding section, we adopted $A_2[f_i, x^{(i)}]^{-1} = (A_2[f_1, x^{(i)}]^{-1}, \ldots, A_2[f_n, x^{(i)}]^{-1})^t$ as the weight vector $w$ to determine the ideal numerical solution in the sense of (6.3). We denote this norm by $\| f \|_{A_2^{-1}}$ and call it simply “normalized norm”. This norm, which differs from the norm (6.4) only by the factor $\epsilon$, is more convenient than (6.4) itself when comparing the computational results in different precisions, because it is independent of the precision of computation. Since $w$ is a vector function of $x^{(i)}$, $w$ is updated as $x^{(i)}$ is updated, so that the norm is different for different stages of iteration. Note that $w$ is not changed in the deceleration step 2).

\textbf{(Stopping rule)}

Stopping rule by $\langle \text{OD} \rangle$: With the ordinary norm, we may determine the solution by stopping the iteration when $\| f \| \leq \delta$ is satisfied, where $\delta$ is some constant given in advance by the user.

Stopping rule by $\langle \text{NN} \rangle$: Since the normalized norm is substantially equivalent to the norm (6.4) (i.e., $\| f \|_{A_2^{-1}} = \epsilon \cdot \| f \|_{3T^{-1}}$), we can easily determine a numerically ideal solution which
satisfies (6.3) by stopping the iteration whenever $\| f \|_{A_1^{-1}}$ is reduced to the order of magnitude of the machine epsilon $\epsilon$.

〈Initial value〉

Initial value 〈AS1〉: A fairly good approximation to the equilibrium state obtained by integrating the ordinary differential equation model of the distillation tower for some time period.

Initial value 〈AS2〉: An approximation obtained by preserving the first three significant digits of each element of the solution vector $\hat{x}$, chopping off the remaining digits.

As has been mentioned before, in terms of 〈NN〉, the condition (6.3) corresponds to $\| f \|_{A_1^{-1}} \leq \epsilon$. The experiment was carried out in single-precision arithmetic and the double-precision on the computer VAX-11/780 (binary, rounding, $\epsilon = 2^{-24}$ for single precision, $\epsilon = 2^{-56}$ for double precision).

![Fig. 8. Process of convergence of the Newton method with deceleration for the 108 functions. (Deceleration coefficients are shown just above the horizontal axis. The upper row is for single precision, and the lower for double precision. $\epsilon_s$ and $\epsilon_d$ beside the vertical axis indicate the machine epsilons for single precision and for double precision.) (a) initial value 〈AS1〉, deceleration by 〈OD〉; (b) initial value 〈AS1〉, deceleration by 〈NN〉; (c) initial value 〈AS2〉, deceleration by 〈OD〉; (d) initial value 〈AS2〉, deceleration by 〈NN〉.](image-url)
(b) Result: For the four cases ((OD & AS1), (NN & AS1), (OD & AS2) and (NN & AS2)), the relations between the number of iterations and the norms $\| f \|$ and $\| f \|_{A_1^{-1}}$ are shown together with deceleration coefficients $\mu$ (when deceleration takes place), in Figs. 8(a)–(d).

(c) Discussion: In the first place, it is observed that in the three cases (NN & AS1), (OD & AS2) and (NN & AS2) (shown in Figs. 8(b)–(d)), the normalized norm $\langle NN \rangle$ reduces to the order of magnitude of the machine epsilon corresponding to the respective precision. It is worth noting that through the behavior of the $\langle NN \rangle$ we can confirm the ideal convergence condition (6.3) is truly satisfied. Furthermore, in the case of double-precision computation, quadratic convergence, a characteristic of the Newton method, is seen clearly through $\langle NN \rangle$.

Among the four cases, the case (OD & AS1) (Fig. 8(a)) failed to find the solution. Figure 8(a) shows that a tragic situation occurs when we use the ordinary norm $\langle OD \rangle$ as the measure for deceleration. That is, deceleration is repeated many times, and what is worse, we cannot determine even whether the sequence has converged or not in this norm. However, in the new norm $\langle NN \rangle$, it is immediately found that the sequence does not yet converge correctly because the value of $\| f \|_{A_1^{-1}}$ is far from the machine epsilon $\epsilon$. This illustrates that how the proposed new norm $\langle NN \rangle$ works effectively in checking convergence, and that it enables us to realize the “automatic convergence check”.

Fig. 8 (continued).
In concluding this section we should make a supplementary remark on the deceleration process by the new norm, in which the weights are changed at each stage of iteration (Step 1).

As we discussed in Section 6, we adopt \( \| f(z) \|_{w(z)} (w(z) = (A_2[z,f_1>, z^{-1}, \ldots, A_2[z,f_n>, z^{-1})') \) as the “merit function” for the deceleration process to determine the \((i+1)\)st approximate \(x^{(i+1)}\) from the \(i\)th approximate \(x^{(i)}\), i.e., we would determine \(x^{(i+1)}\) so that the condition

\[
\| f(x^{(i+1)}) \|_{w(x^{(i+1)})} = \| f(x^{(i)} + \mu \Delta x^{(i)}) \|_{w(x^{(i)})} \\
\leq (1 - \frac{1}{2}\mu) \| f(x^{(i)}) \|_{w(x^{(i)})}
\]

might be satisfied by choosing some finite \(\mu > 0\).

However, since \(w(z)\) as well as \(f(z)\) is a function of the point \(z\), the ordinary Newton direction is not always a descent direction of the merit function \(\| f(z) \|_{w(z)}\). Hence, to guarantee that the deceleration condition is satisfied with some “finite” step-size \(\mu\) without fail, we must adopt the “norm” with the weights fixed at \(x^{(i)}\), i.e., \(w(x^{(i)})\), throughout a deceleration step, and determine \(x^{(i+1)}\) in such a way that the inequality

\[
\| f(x^{(i+1)}) \|_{w(x^{(i)})} = \| f(x^{(i)} + \mu \Delta x^{(i)}) \|_{w(x^{(i)})} \\
\leq (1 - \frac{1}{2}\mu) \| f(x^{(i)}) \|_{w(x^{(i)})}
\]

may hold. This is a kind of tangent-space viewpoint, and may not give a satisfactory result if the weights change very rapidly near \(x^{(i)}\), so that \(\| f(x^{(i)}) \|_{w(x^{(i)})}\) will not be expected to be monotone decreasing with \(i\).

Fortunately, the deceleration procedure worked well for all the cases we have tested. However, it is an important subject of research to develop a theoretically better founded method of deceleration such that the convergence is guaranteed even when the weights of the norm change rapidly.

8. Metrics induced by the rounding errors

In Sections 6 and 7 we took the \(n\)-dimensional hypercube

\[
[f_1(x) - \Delta f_1(x), f_1(x) + \Delta f_1(x)] \times \cdots \times [f_n(x) - \Delta f_n(x), f_n(x) + \Delta f_n(x)]
\]

for that area in which the exact values of \(f(x)\) may lie, where \(f_i(x)\) and \(\Delta f_i(x)\) are the computed value of the function \(f_i(x)\) at \(x\) and the sharp upper bound for the rounding error in it. However, we can investigate the shape of the region in which the exact values of \(f(x)\) lie more precisely by assuming that the probabilistic model of rounding errors in Section 3 works well. In that case the covariance matrix \([\sigma^2]\) of the rounding error \(\Delta f\) of the function \(f\), the \((i, j)\)th element of which is defined to be the covariance of the rounding errors of \(f_i\) and \(f_j\) by (3.23), gives meaningful information on the distribution of the rounding errors. Thus, the distribution of the rounding errors may be represented by the hyperellipsoid

\[
H(x) = \left\{ (f_1(x) + \Delta f_1, \ldots, f_n(x) + \Delta f_n) \mid \sum_{i,j} [\sigma^2(x)]_{ij}^{-1} \Delta f_i \Delta f_j \leq 1 \right\}.
\]

Since \(\sigma^2(x)\) depends on \(f\) as well as on \(x\), the matrix \([\sigma^2]^{-1}\) may be considered as a Riemann metric in the space of \(f\), i.e., in the space of residuals.
If we map the hyperellipsoid into the space of variables $x$ by using the Jacobian matrix, we obtain a hyperellipsoid in the space

$$S(x) = \left\{ (x_1 + \Delta x_1, \ldots, x_n + \Delta x_n) \mid \sum_{i,j} \tau_{ij} \Delta x_i \Delta x_j \leq 1 \right\},$$

(8.3)

where

$$\tau_{ij} = \sum_{k,l} \left[ \sigma^2 \right]^{-1}_{k,l} J_{ki} J_{lj}.$$  \hspace{1cm} (8.4)

It is obvious that, if we take for $x$ the numerical solution $\hat{x}$ of the equation $f(x) = 0$, this hyperellipsoid may be interpreted as the region in which the exact solution of the equation lies. Also we may regard $\tau$ as a Riemann metric in the space of variables.

In this way, we can introduce the two natural metrics, $[\sigma^2]^{-1}$ in the space of residuals and $\tau$ in the space of variables.

9. Conclusion

The problem of defining a practically more reasonable norm in solving a system of nonlinear equations has been taken up as an application of the fast differentiation algorithm.

The algorithm was applied to solve a large-scale system of nonlinear equations with 108 variables arising in a model of a distillation tower in a chemical plant, and the following advantages of the method were confirmed through the computational experiments:

(1) This algorithm computes the accurate Jacobian matrix 6 times faster than the numerical differentiation;
(2) Fairly sharp estimates of rounding errors of the functions can be obtained in a practicable time;
(3) The new norm works nicely and is of fundamental importance in observing convergence.

In addition, it was shown that meaningful metrics in the space of residuals and that of variables can be defined by means of the rounding error estimates.

Recently the authors and K.V. Kim et al. in the Soviet Union [10,11,12] have developed, independently of one another, the new method for computing the product of a vector and the Hessian matrix of a function or the product of a vector and the Jacobian matrix of a system of functions with a complexity proportional to that of computing the function or the system of the functions. Those methods are applicable in particular to the ordinary differential equations or the optimization problems. Investigation in that direction including more efficient implementation of the fast differentiation algorithm is now being carried on [6,7].

Acknowledgements

The authors thank Mr. Junkichi Tsunekawa (presently, Professor of Teikyo University of Technology) and his colleagues in the Union of Japanese Scientists and Engineers for providing them with the data of the system of nonlinear equations of the distillation tower based on the DPS model and for their valuable advices. They are also indebted to Professor Hiroshi Imai of Kyushu University for reading the manuscript and giving them valuable advices.
Appendix

In this paper we proposed two types of the estimates of rounding errors. The difference between the two is how to estimate the upper bound of the generated rounding error $\max( |\delta v_i| )$

---

Fig. A.1. Comparison of the observed maximum rounding errors $\max |\Delta f_i|$ (among 100 samples) with the theoretical estimates $A_i[f_i, x] \epsilon$ and $P_i[f_i, x] \epsilon$ for the 108 functions. (Dotted line: observed value = estimate×10; solid line: observed value = estimate; broken line: observed value = estimate/10; chained line: observed value = estimate/100.)

(a) VAX-11/780, absolute bound (3.10). (b) M-280H, absolute bound (3.10). (c) VAX-11/780, probabilistic bound (3.17). (d) M-280H, probabilistic bound (3.21).
of each intermediate variable $v_i$. In the main part of this paper we used the estimates $A_2 \cdot \epsilon$ and $P_2 \cdot \epsilon$ ((3.11) and (3.18)), where the rougher estimate $|v_i| \cdot \epsilon$ for $\max(|\delta v_i|)$ was adopted, chiefly for the sake of the simplicity of computation. In this appendix, we shall examine the latter type of estimates $A_1 \cdot \epsilon$ and $P_1 \cdot \epsilon$, where the sharper bound $m(v_i) \cdot \epsilon$ for $\max(|\delta v_i|)$ is used. In the following, $\beta = 2$ or $16$ denotes the base of the floating-point representation.

We investigate the property of the estimates $A_1 \cdot \epsilon$ of (3.10) and $P_1 \cdot \epsilon$ of (3.17) (in the case of rounding) or (3.21) (in the case of chopping) by the same experiment as in section 5.2 (under the same conditions and samples). The results are shown in Fig. A.1 (the relation between the observed maximum rounding errors and the estimates) and Fig. A.2 (the degree of overestimate).

1. Absolute bound: As seen from Fig. A.2(a), in the case of VAX-11/780 (binary, rounding, $\epsilon = 2^{-24}$ for single precision), the absolute bounds are 2.7 times as large as the observed maximum rounding errors on the average (5 times at the maximum). As for M-280H (hexadecimal, chopping, $\epsilon = 16^{-5}$ for single precision), the absolute bounds are about 3.3 times on the average (10 times at the maximum) (Fig. A.2(b)).

2. Probabilistic bound: The ratios of the observed maximum rounding errors to their estimates are within the range of 1.7–3.7 in the case of VAX-11/780 (Fig. A.2(c), and 1.4–3.2 in the case of M-280H (Fig. A.2(d)). This shows that in both cases the estimates are almost exact as the estimate of the standard deviation or the square root of mean square error, respectively.

Thus, it is observed that the sharpness of the estimate is fairly improved by using $m(v_i) \cdot \epsilon$ compared with the case of Section 5. In particular in the case of $\beta = 16$, the estimates $A_1 \cdot \epsilon$ and $P_1 \cdot \epsilon$ give bounds better by a factor of 2 or 3 compared with $A_2 \cdot \epsilon$ and $P_2 \cdot \epsilon$.

As discussed in Section 3, $m(v_i) \cdot \epsilon$ can be sharper than $|v_i| \cdot \epsilon$ by a factor of $\beta$ in the best case (cf. (3.8)). Hence $A_1 \cdot \epsilon$ and $P_1 \cdot \epsilon$ could also give better bounds by a factor of 16 in the case of the hexadecimal computation and by a factor of 2 in the case of binary computation than...
$A_2 \cdot \epsilon$ and $P_2 \cdot \epsilon$ in Section 5. We can clearly confirm this tendency through the result of this experiment.

Furthermore, it is because $m(v_i) \cdot \epsilon$ is an almost exact estimate of $\max(|\delta v_i|)$ that the sharpness of the estimates by $A_1 \cdot \epsilon$ and $P_1 \cdot \epsilon$ does not depend on $\beta$ very much, as is observed by the experiment.

If we prepare a special subroutine for computing $m(v)$ from $v$ (preferably written in an assembler language), it is not very costly to compute $m(v_i)$ each time when the intermediate variable $v_i$ is computed. Hence, it is recommended to use the estimates $A_1 \cdot \epsilon$ and $P_1 \cdot \epsilon$ based on $m(v_i) \cdot \epsilon$, in particular, in the case of hexadecimal computation.

References