BIT 16 (1976), 146–160

TAYLOR EXPANSION OF THE ACCUMULATED ROUNDING ERROR

SEPPO LINNAINMAA

Abstract.

The article describes analytic and algorithmic methods for determining the coefficients of the Taylor expansion of an accumulated rounding error with respect to the local rounding errors, and hence determining the influence of the local errors on the accumulated error. Second and higher order coefficients are also discussed, and some possible methods of reducing the extensive storage requirements are analyzed.

1. Introduction.

When computation is carried out with finite precision, the resulting value of a numerical process is in general different from the true value of the result. The difference between these two values is called the accumulated rounding error of the result. It is caused by the errors in the initial values and by the local rounding errors which occur in each intermediate operation. Generally the accumulated rounding error can be expanded as a Taylor expansion with respect to the local errors, and thus the effect of the local errors upon the accumulated error can be found, as noted by Henrici [1]. In this article analytic and algorithmic methods are presented for this purpose. The methods are a further development of those given in [2] and [6].

Usually the accumulated error can be approximated quite accurately using only the first order terms of the Taylor expansion [1], [5], [3]. Thus the emphasis in this article is on determination of the first order coefficients, although higher orders are also considered.

For the *a posteriori* algorithmic computation of the coefficients of the Taylor expansion it is necessary to save some information on all the operations of the process. The amount of computer storage required can be prohibitively large, and some methods of reducing it are discussed.

Received Jan. 22, 1976.

2. The Taylor expansion of the rounding errors.

A computing process usually consists of a sequence of real numbers, say $u_1, u_2, \ldots, u_N$. Some numbers in this sequence are initial values, the others being produced by operations whose operands (usually two) appear earlier in the sequence, i.e.

$$u_i = Q_i(u_j, u_k), \quad j,k<i\ , \tag{1}$$

where $Q_i(u_j,u_k)$ may denote e.g. $-u_j$, u_j+u_k, u_j-u_k, $u_j \times u_k$ or u_j/u_k.

In actual computations, floating-point numbers u_i' of some finite precision t are normally used instead of the non-terminating real numbers u_i. Thus, if the value of a real number u rounded to t digits is denoted by $\mathrm{fl}(u,t)$, the operation (1) is replaced by a floating-point operation

$$u_i' = \mathrm{fl}\big(Q_i(u_j', u_k'), t\big)\ , \tag{2}$$

i.e. each accurate result is rounded to t digits. The *local (absolute) rounding error* $r(u_i)$ (or r_i) of u_i is defined as

$$r_i = r(u_i) = u_i' - Q_i(u_j', u_k')\ . \tag{3}$$

In some connections it is more useful to consider the local *relative* rounding error

$$\hat{r}_i = \hat{r}(u_i) = r_i/Q_i(u_j', u_k')\ . \tag{4}$$

If the whole process contained no rounding errors, then u_i' would be replaced by u_i, $i=1,\ldots,n$. The *accumulated rounding error* $R(u_i)$ (or R_i) of u_i is the difference of these two numbers, i.e.

$$R_i = R(u_i) = u_i' - u_i\ . \tag{5}$$

Using (1) and (3), (5) can be expressed as

$$R_i = r_i + Q_i(u_j', u_k') - Q_i(u_j, u_k)$$

and hence, using the Taylor expansion, as

$$R_i = r_i + \sum_{n=1}^{\infty} \sum_{m=0}^{n} \big\{(m!\,(n-m)!\,)^{-1} \big(\partial^n Q_i(x,y)/\partial x^m \partial y^{n-m}\big)_{x=u_j;\, y=u_k}\big\} R_j{}^m R_k{}^{n-m} \tag{6}$$

where R_j and R_k are defined by (5). Below the values of the coefficients of (6) in braces are denoted by the abbreviated notations $d_{i,j}$, $d_{i,k}$, $d_{i,jj}$, $d_{i,jk}$ and $d_{i,kk}$ respectively for the value pairs $(1,0)$, $(0,1)$, $(2,0)$, $(1,1)$ and $(0,2)$ of $(m,n-m)$. Thus, for instance,

$$d_{i,jk} = \partial^2 Q_i(x,y)/(\partial x\, \partial y)|_{x=u_j,\, y=u_k}\ .$$

With these notations, (6) can be written as

$$(7) \quad R_i = r_i + d_{i.j}R_j + d_{i.k}R_k + d_{i.jj}R_j^2 + d_{i.jk}R_jR_k + d_{i.kk}R_k^2 + O(R^3) ,$$

where $O(R^3)$ stands for terms of the third and higher orders with respect to the R's. For example, if $Q_i(x,y)$ denotes $x \times y$, then (7) is equal to $R_i = r_i + u_kR_j + u_jR_k + R_jR_k$.

Table 1. *Coefficients of 1st and 2nd order terms of (7) for elementary operations* $Q_i(u_j, u_k)$.

$Q_i(u_j,u_k)$	$d_{i.j}$	$d_{i.k}$	$d_{i.jj}$	$d_{i.jk}$	$d_{i.kk}$
$-u_j$	-1	0	0	0	0
u_j+u_k	1	1	0	0	0
u_j-u_k	1	-1	0	0	0
$u_j \times u_k$	u_k	u_j	0	1	0
$u_j \,/\, u_k$	$1/u_k$	$-u_j/u_k^2$	0	$-1/u_k^2$	u_j/u_k^3

The values of some of the coefficients d for elementary operations Q_i are listed in Table 1. The coefficients of the higher order terms are zero for the operations given in Table 1 except for division, in which case the coefficients of ${R_k}^n$, $R_j{R_k}^{n-1}$ and ${R_j}^m{R_k}^{n-m}$, $m>1$, are respectively $-u_j(-u_k)^{-n-1}$, $-(-u_k)^{-n}$ and 0, $n=1,2,3,\ldots$. As well as those operations given in Table 1 some other operations can be treated as elementary operations in the sense that they are connected with a single local error. For instance, if $Q_i(u_j,u_k)=\ln(u_j)$ then the coefficient of ${R_j}^n$ in (7), $n=1,2,3,\ldots$, is $-(-u_j)^{-n}$.

Expressions similar to those for R_i can also be otained for R_j and R_k in terms of r_j, r_k and the accumulated rounding errors of the operands which produced u_j and u_k. Since $R_i=r_i$ for an initial value u_i, the following expansion is obtained for the accumulated rounding error R_i after a finite number of steps:

$$(8) \qquad R_i = \sum_{p=1}^{i} c_{i.p}r_p + \sum_{p=1}^{i}\sum_{q=1}^{p} c_{i.pq}r_pr_q + O(r^3)$$

where the c's are some scalar coefficients and $O(r^3)$ stands for terms of the third and higher orders with respect to the r's.

The expansion (8) is given in terms of the aboslute errors r_i. Naturally, a corresponding expansion exists in terms of the relative errors. Below we analyze the relationship between the coefficients of these two expansions.

Formulae (3) and (4) imply $\hat{r}_i = r_i/(u_i' - r_i)$. Hence, since according to (8) u_i' can be written as $u_i + \sum_{p=1}^{i} c_{i.p}r_p + O(r^2)$,

$$r_i = u_i\hat{r}_i + \sum_{p=1}^{i-1} c_{i.p}u_p\hat{r}_i\hat{r}_p + O(\hat{r}^3) .$$

On substitution into (8) this yields

$$R_i = \sum_{p=1}^{i} (u_p c_{i.p})\hat{r}_p + \sum_{p=1}^{i} (u_p{}^2 c_{i.pp})\hat{r}_p{}^2 + \sum_{p=1}^{i} \sum_{q=1}^{p-1} u_q(u_p c_{i.pq} + c_{i.p}c_{p.q})\hat{r}_p\hat{r}_q + O(\hat{r}^3) \,. \tag{9}$$

As can be seen, for coefficients of the first order only a simple modification is needed when relative errors are used rather than absolute ones. The connection between the coefficients becomes more complicated the higher the order. The following is restricted to the consideration of absolute errors.

It is worth noting that if u_p is an initial value, then the coefficient $c_{i.p}$ in (8) is *not* dependent on the algebraic sequence which has been used to produce u_i. In fact, due to the linearity of (3), each coefficient $c_{i.p}$ is the partial derivative of u_i with respect to u_p. More generally, the coefficient c of $r_{p_1}^{n_1} r_{p_2}^{n_2} \ldots r_{p_m}^{n_m}$ in (8), where $u_{p_1}, \ldots, u_{p_m}$ are initial values, is equal to

$$c = (n_1!\, n_2! \ldots n_m!)^{-1} \partial^{n_1+n_2+\ldots+n_m} u_i / \partial u_{p_1}^{n_1} \partial u_{p_2}^{n_2} \ldots \partial u_{p_m}^{n_m} \,. \tag{10}$$

For example, if $w = u^2 - v^2$, then the coefficients of $r(u)$ and $r(u)^2$ in the expansion (8) of $R(w)$ are $2u$ and 1, respectively, independent of whether w is computed by $(u \times u) - (v \times v)$ or by $(u+v) \times (u-v)$.

If we denote the sum of the nth order terms in (8) by $R_i^{(n)}$ then (8) can be written as

$$R_i = \sum_{n=1}^{\infty} R_i^{(n)} \,. \tag{11}$$

Correspondingly, (7) can be fractioned in the following parts:

$$R_i^{(1)} = r_i + d_{i.j} R_j^{(1)} + d_{i.k} R_k^{(1)} \,, \tag{12a}$$

$$R_i^{(2)} = d_{i.j} R_j^{(2)} + d_{i.k} R_k^{(2)} + d_{i.jj} (R_j^{(1)})^2 + d_{i.jk} R_j^{(1)} R_k^{(1)} + d_{i.kk} (R_k^{(1)})^2 \,, \tag{12b}$$

$$\ldots$$

As will be seen in the following sections, the formulae (12) can be used effectively in the determination of the coefficients c of (8).

3. The analytic determination of the Taylor coefficients.

For fairly simple algorithms the coefficients of (8) can be determined analytically using the recursion formula (7). This yields a difference equation which can be solved if it is not too complicated [1]. As an example, consider the calculation of the value of a polynomial using Horner's scheme.

The value of the polynomial

$$w_n = a_0 x^n + a_1 x^{n-1} + \ldots + a_n \tag{13}$$

is computed using Horner's scheme as

$$\text{(14)} \qquad \begin{cases} w_0 = a_0 , \\ v_i = x \times w_{i-1}, \; w_i = a_i + v_i, \quad i = 1, \ldots, n . \end{cases}$$

The application of (12) to (14) yields

$$\text{(15a)} \qquad R(w_0)^{(1)} = r(w_0) + r(a_0) + r(v_0) ,$$

$$\text{(15b)} \qquad R(v_i)^{(1)} = r(v_i) + w_{i-1} R(x)^{(1)} + x R(w_{i-1})^{(1)} ,$$

$$\text{(15c)} \qquad R(w_i)^{(1)} = r(w_i) + R(a_i)^{(1)} + R(v_i)^{(1)} ,$$

where the zero-valued rounding errors $r(w_0)$ and $r(v_0)$ are included for simplifying indexing in the following formulae. Since x and the coefficients a_i are initial values we have $R(x)^{(1)} = r(x)$ and $R(a_i)^{(1)} = r(a_i)$, $i = 0, \ldots, n$. Thus, substituting (15b) for (15c) we obtain the difference equation

$$\text{(16)} \qquad R(w_i)^{(1)} = r(w_i) + r(a_i) + r(v_i) + w_{i-1} r(x) + x R(w_{i-1})^{(1)}$$

with the initial condition $R(w_0)^{(1)} = r(w_0) + r(a_0) + r(v_0)$. The solution of (16) is

$$\text{(17)} \qquad R(w_i)^{(1)} = \sum_{j=0}^{i} x^{i-j} \big(r(w_j) + r(a_j) + r(v_j) \big) + \sum_{j=1}^{i} x^{i-j} w_{j-1} r(x) , \quad i = 0, \ldots, n .$$

From (14)

$$\text{(18)} \qquad w_i = \sum_{j=0}^{i} a_j x^{i-j} ,$$

and thus (17) may be written as

$$\text{(19)} \qquad R(w_i)^{(1)} = \sum_{j=0}^{i} x^{i-j} \big(r(w_j) + r(a_j) + r(v_j) \big) + \sum_{j=0}^{i-1} (i-j) a_j x^{i-j-1} r(x) .$$

In particular, the coefficient of $r(x)$ is seen to be equal to dw_n/dx when $i = n$ and w_n is as in (13). This is in accordance with (10).

The higher order terms of the Taylor expansion can be determined in a similar way. Utilizing (12) the formulae

$$\text{(20)} \qquad \begin{cases} R(w_0)^{(m)} = 0 , \\ R(v_i)^{(m)} = w_{i-1} R(x)^{(m)} + x R(w_{i-1})^{(m)} + \sum_{p=1}^{m-1} R(x)^{(p)} R(w_{i-1})^{(m-p)} , \\ R(w_i)^{(m)} = R(a_i)^{(m)} + R(v_i)^{(m)}, \quad m > 1 , \end{cases}$$

are obtained for the mth order terms. Since x and the a_i's are initial values, $R(x)^{(m)} = R(a_i)^{(m)} = 0$, $i = 0, \ldots, n$, for $m > 1$. Thus the difference equation

$$\text{(21)} \qquad R(w_i)^{(m)} = x R(w_{i-1})^{(m)} + r(x) R(w_{i-1})^{(m-1)}$$

is obtained with the initial condition $R(w_0)^{(m)} = 0$. The solution of (21) is

$$\text{(22)} \qquad R(w_i)^{(m)} = \sum_{p=1}^{i} x^{i-p} r(x) R(w_{p-1})^{(m-1)}, \quad i = 1, \ldots, n .$$

With the aid of (19) and (18), the recursive use of (22) yields

$$(23)\qquad R(w_i)^{(m)} = \sum_{p=0}^{i-m+1}\binom{i-p}{m-1}x^{i-p-m+1}\big(r(w_p)+r(a_p)+r(v_p)\big)r(x)^{m-1}$$
$$+\sum_{p=0}^{i-m}\binom{i-p}{m}a_p x^{i-p-m}r(x)^m,\ n\geqq 1,\quad i=0,\ldots,n\,.$$

The highest order non-zero term of $R(w_n)$, $n\geqq 0$, is $r(a_0)r(x)^n$.

The above example shows how formula (7) can be used to determine the Taylor coefficients analytically. In the special case of Horner's scheme these coefficients can in fact be derived in a far simpler way. Directly from (10) and (18) it follows that the coefficient of $r(x)^m$ in the Taylor expansion of w_i is equal to

$$(m!)^{-1}\ \partial^m w_i/\partial x^m = \sum_{p=0}^{i-m}\binom{i-p}{m}a_p x^{i-p-m},\quad m=1,\ldots,i\,,$$

and the coefficient of $r(a_p)r(x)^{m-1}$, $p=0,\ldots,i-1$, $m=1,\ldots,i$, is equal to

$$\big(1/(m-1)!\big)\ \partial^m w_i/\partial a_p\partial x^{m-1} = \binom{i-p}{m-1}x^{i-p-m+1}\,,$$

agreeing with the coefficients given in (23). Further, the coefficients of $r(a_p)^q r(x)^m$ are obviously zero when $q>1$, and (20) clearly indicates that the coefficients of $r(w_p)r(x)^m$ and $r(v_p)r(x)^m$ are equal to the coefficient of $r(a_p)r(x)^m$.

4. The algorithmic determination of the Taylor coefficients.

The algorithmic *a posteriori* determination of the coefficients c of (8) is based on the same principles as the analytic determination. When the first order coefficients of R_n are determined, the process proceeds in n cycles. Auxiliary coefficents $c_p^{[i]}$, $p=1,\ldots,n$, are computed during the $(n-i)$th cycle. These coefficients are determined so that the identity

$$(24)\qquad R_n^{(1)} = \sum_{p=1}^{i} c_p^{[i]}R_p^{(1)}+\sum_{p=i+1}^{n} c_p^{[i]}r_p$$

remains true throughout the execution of the determination process. Initially, i.e. with $n-i=0$, this is established by setting $c_n^{[n]}=1$, $c_p^{[n]}=0$, $p\neq n$, from which the trivial identity $R_n^{(1)}=R_n^{(1)}$ follows. Generally, suppose that (24) is true after the $(n-i)$th cycle, $i=n,n-1,\ldots,1$, and that the operands producing u_i were u_j and u_k. Then substituting (12a) for (24) yields

$$R_n^{(1)} = \sum_{p=1}^{i-1} c_p^{[i]}R_p^{(1)}+c_i^{[i]}r_i+c_i^{[i]}d_{i,j}R_j^{(1)}+c_i^{[i]}d_{i,k}R_k^{(1)}+\sum_{p=i+1}^{n} c_p^{[i]}r_p\,.$$

Setting $c_j^{[i-1]}\leftarrow c_j^{[i]}+c_i^{[i]}d_{i,j}$, $c_k^{[i-1]}\leftarrow c_k^{[i]}+c_i^{[i]}d_{i,k}$ and letting the other

$c_p^{[i-1]}$'s be identical to the corresponding $c_p^{[i]}$'s, it is seen that (24) remains true after the $(n-i+1)$'th cycle. Finally, after the nth cycle, the coefficient $c_p^{[0]}$ is obviously identical with the Taylor coefficient $c_{n.p}$, $p=1,\ldots,n$ due to the uniqueness of the Taylor expansion.

The determination of the coefficients declared above requires information to be saved from each operation. The amount of this information is so great that details of the actual use of computer storage have been included in the algorithmic presentation of the process described above. This is to show that external files can be used in saving the extra information instead of the main storage, thus permitting the consideration of large numerical processes [6].

Each number u_p is saved as the contents of a computer storage cell $z_i \in \{z_1, z_2, \ldots, z_m\}$. The contents may be changed one or more times during the computing process. In computer programming, the operation producing u_p is denoted by referring to the cells z_j and z_k which contain the associated operands. The execution of Algorithm T (given below) presupposes that on each such operation the indices i, j and k as well as the partial derivatives of the result u_p (saved in z_i; note that in Algorithm T i, j and k are indices of storage cells, *not* subscripts of numbers as in (24)) with respect to the operands (contained in z_j and z_k) are saved respectively as the values of $I[p]$, $J[p]$, $K[p]$, $Dj[p]$ and $Dk[p]$. I, J, K, Dj and Dk are tables placed in an external file. In cases where only one operand exists or where there are no operands, the corresponding values of $Dj[p]$ and $Dk[p]$ are set to zero. When the value u_n has been computed, its Taylor coefficients can be determined using Algorithm T, which utilizes the contents of the previously mentioned tables in reverse order.

Algorithm T (Determination of the first order coefficients of the Taylor expansion of the accumulated rounding errors). Initially the table elements $I[p]$, $J[p]$, $K[p]$, $Dj[p]$ and $Dk[p]$, $p=1,\ldots,n$, contain respectively the indices of the computer storage cells in which each number u_p belonging to the actual computing process has been saved together with the values of its first and second operands, and the first order partial derivatives of u_p with respect to these operands. The algorithm produces the first order coefficients $c_{n.p}$ of the Taylor expansion of the accumulated rounding error of u_n. It is assumed that u_n is the contents of z_N when the algorithm T is called, and that m computer storage cells $z_1, z_2, \ldots, z_m$ have been utilized during the numerical process. Algorithm T utilizes also the table $C[1],\ldots,C[m]$ for saving the values of the requisite coefficients of (24); the value of $c_p^{[i]}$ is saved in $C[I[p]]$.

$T1$. [Initialize.] $p \leftarrow n$, $C[i] \leftarrow 0$ for $i = 1,\ldots,m$, $i \neq N$, $C[N] \leftarrow 1$.

$T2$. [Read.] $i \leftarrow I[p]$, $j \leftarrow J[p]$, $k \leftarrow K[p]$, $dj \leftarrow Dj[p]$, $dk \leftarrow Dk[p]$.

$T3$. [Coefficient $c_{n,p}$ completed.] coef $\leftarrow C[i]$, $C[i] \leftarrow 0$. The contents of coef is now equal to the value of $c_{n,p}$ and can be utilized.

$T4$. [Coefficient zero?] If coef $= 0$, go to step $T6$.

$T5$. [Update table C.] If $dj \neq 0$ then $C[j] \leftarrow C[j] + \text{coef} \times dj$. If $dk \neq 0$ then $C[k] \leftarrow C[k] + \text{coef} \times dk$.

$T6$. [Decrease p.] Decrease p by 1. If $p > 0$, return to step $T2$, otherwise the algorithm terminates.

It should be noted that Algorithm T in fact uses the rounded numbers u_p' instead of the numbers u_p in actual computing. Rounding errors also occur in the computation of Algorithm T itself.

Instead of the values $Dj[p]$ and $Dk[p]$, the *preceding* contents of z_i and the type of the operation producing u_p may be saved. If this is done Algorithm T must be modified so that the table $Z[1],\ldots,Z[m]$ initially contains the final values of $z_1,\ldots,z_m$. In step $T2$ the modified information is read, the value of $Z[i]$ being replaced by the „preceding" contents of z_i. The values of dj and dk are then computed utilizing Table 1, the type of the operation just read, and the contents of $Z[j]$ and $Z[k]$ respectively as the values of u_j and u_k. No other changes are required.

Coefficients of higher orders can also be determined algorithmically, but the required algorithm becomes more complicated and wasteful of storage the higher the order. For instance, for the second order coefficients the equation corresponding to (24) is

$$R_n^{(2)} = \sum_{p=1}^{i} c_p^{[i]} R_p^{(2)} + \sum_{p=1}^{i} \sum_{q=1}^{p} h_{pq}^{[i]} R_p^{(1)} R_q^{(1)} + \sum_{p=i+1}^{n} \sum_{q=1}^{i} h_{pq}^{[i]} r_p R_q^{(1)} + \sum_{p=i+1}^{n} \sum_{q=i+1}^{p} h_{pq}^{[i]} r_p r_q \,. \tag{25}$$

Clearly an algorithm for computing the second order terms is essentially more complicated than Algorithm T. If, for simplicity, each u_i is supposed to be located in a separate cell, then the unoptimized Algorithm S given below is obtained.

Algorithm S (Determination of the first and second order Taylor coefficients). Initially the table elements $U[i]$, $J[i]$, $K[i]$ and $Q[i]$, $i = 1,\ldots,n$, contain respectively the indices of the first and second operands of u_i, the values of u_i and the type of the operation which produced u_i. The algorithm produces the first and second order coefficients $c_{n,i}$ and $c_{n,ij}$ of the Taylor expansion of the accumulated error of u_n. Algorithm S utilizes tables $C[i]$ and $H[i,j]$, $i = 1,\ldots,n$, $j = 1,\ldots,i$, for saving the values of the coefficients of (25). When the algorithm

terminates, $C[i]$ contains the coefficient $c_{n.i}$ and $H[i,j]$ the coefficient $c_{n.ij}$, $i=1,\ldots,n$, $j=1,\ldots,i$. Note that H is a lower triangular matrix. Due to the simplified formal presentation, in the following algorithm the „illegal" notation $H[i,j]$ where $i<j$ actually refers to the element $H[j,i]$.

*S*1. [Initialize.] $C[n] \leftarrow 1$, $C[i] \leftarrow 0$ for $i=1,\ldots,n-1$, $H[i,j] \leftarrow 0$ for $i=1,\ldots,n$, $j=1,\ldots,i$, $i \leftarrow n$.

*S*2. [Prepare for updating.] $j \leftarrow J[i]$, $k \leftarrow K[i]$. Compute the values of dj, dk, djj, djk and dkk representing respectively the entries $d_{i.j}$, $d_{i.k}$, $d_{i.jj}$, $d_{i.jk}$ and $d_{i.kk}$ of Table 1, where the type of the operation is given by $Q[i]$ and the values of u_j and u_k are given respectively as the contents of $U[j]$ and $U[k]$.

*S*3. [Update the coefficients.] $C[j] \leftarrow C[j]+C[i]\times dj$, $C[k] \leftarrow C[k]+C[i]\times dk$,
$H[j,s] \leftarrow H[j,s]+H[i,s]\times dj$, $H[k,s] \leftarrow H[k,s]+H[i,s]\times dk$ for $s=1,\ldots,n$, $s \neq i$,
$H[j,j] \leftarrow H[j,j]+C[i]\times djj+H[i,i]\times dj\times dj$,
$H[k,k] \leftarrow H[k,k]+C[i]\times dkk+H[i,i]\times dk\times dk$,
$H[j,k] \leftarrow H[j,k]+C[i]\times djk+2\times H[i,i]\times dj\times dk$,
$H[i,j] \leftarrow H[i,j]+2\times H[i,i]\times dj$, $H[i,k] \leftarrow H[i,k]+2\times H[i,i]\times dk$.

*S*4. [Decrease i.] Decrease i by 1. If $i>0$, return to step *S*2, otherwise the algorithm terminates.

5. Storage and time requirements for the algorithmic determination of the Taylor coefficients.

5.1. *Storage requirements.*

As found in section 4, the execution of Algorithm T for determining the first order coefficients presupposes that five numbers are saved for each of the n operations of the computing process. External storage can be used for this purpose, and it may be of serial access type provided that it can be read in reverse order.

Additionally, m main storage cells are needed for saving the requisite coefficients of (24), m being the number of main storage cells utilized by the actual computing process. The number of auxiliary variables is quite negligible, and thus the execution of Algorithm T requires roughly $5n$ *external and* m *internal storage cells*. Usually the computing process does not utilize m consequtive indexed cells z_i. Thus m additional "reference cells" are required for an implementation using a high-level language, each containing the "index" of a variable [4]. The modification

mentioned after Algorithm T requires a further m internal storage cells. If the n completed coefficients are to be saved, n more cells are required. Through packing more than one information unit in each cell more effective storage use can be obtained, but the order of magnitude remains unchanged.

In many applications, e.g. in approximating the mean value and standard deviation of the accumulated rounding error [3], [4], some function of the Taylor coefficients is required rather than the coefficients themselves. In such cases the extra storage required can be reduced to some extent. Possibilities for doing this are analyzed in section 6.

When the coefficients $c_{i,p}$ are required only for some fixed p, expressing the effect of a single local error r_p on the values u_i, then (8) has only one term. Obviously (12a) can then be applied simultaneously with the computing process, and only m extra storage cells are required for saving the coefficients.

If coefficients of second and higher orders are required then the need for external storage is not increased, but for the computation of the hth order coefficients the number of main storage cells required is roughly proportional to mn^{h-1}. Simplified algorithms such as Algorithm S have a requirement for main storage cells which is roughly proportional to n^h.

5.2. *Time requirements.*

For use in analyzing the time required for the execution of Algorithm T, the arithmetic operations required to compute the first order partial derivatives, given in Table 1, are listed in Table 2. Account is taken of the fact that $-u_j/u_k^2 = -d_{i,j} \times u_i$. The arithmetic operations executed for each original operation during steps $T5$ and $T6$ of Algorithm T are also listed, noting specifically the special cases where the absolute value of a partial derivative is equal to 1.

Table 2. *The number of arithmetic operations required in the execution of Algorithm T, when the ones of Table* 1 *are utilized in step T5.*

$Q_i(u_j, u_k)$	partial deriv.			step $T5$			step $T6$
	neg.	mult.	div.	add.	sub.	mult.	sub.
$-u_j$	0	0	0	0	1	0	1
$u_j + u_k$	0	0	0	2	0	0	1
$u_j - u_k$	0	0	0	1	1	0	1
$u_j \times u_k$	0	0	0	2	0	2	1
u_j/u_k	1	1	1	2	0	2	1

On the basis of Table 2, noting the possible jump directly from step $T4$ to $T6$, Algorithm T might be expected to be quite effective. Unfortunately the work is not restricted to arithmetic operations. As noted in [6], due to effective buffering the time required by I/O operations is not dominant, but there also exist numerous logical operations, replacement operations, and index determinations and searches. Thus the following experimental result is not surprising. Algorithm T was programmed for a Burroughs B6700 computer using Algol. The value of a thousandth degree polynomial with randomly chosen coefficients was computed using Horner's scheme, both with and without determination of the first order Taylor coefficients. The ratio between the times was about 20. Possibly by using a lower level language this ratio could be halved. In most applications the ratio would also be lower, since arithmetic floating-point operations are not generally as dominant.

As is clearly seen from Table 2, if Algorithm T is used to obtain the Taylor coefficients of several intermediate values of the same computing process, then it is more time-effective to compute the partial derivatives during the process as proposed than to include their computation in Algorithm T as done in the modification mentioned after Algorithm T. This is especially true when division occurs frequently, since the derivatives must then be actually computed rather than merely found from a table.

6. The computing process as a graph.

Any computing process can be represented as a directed graph in which each vertex is associated with a number belonging to the process. Below the vertex associated with number u_i will be called the vertex u_i. An arc exists from u_j to u_i if and only if u_j is an operand of the operation which produces u_i. In particular, if u_i is an initial value then no arcs lead to the vertex u_i. Obviously the graph grows as the computing process proceeds. A characteristic feature of the graph of any computing process is that it contains no oriented cycles.

Each arc (from u_j to u_i) is associated with a number $d_{i,j}$ as defined in (7), which may be called the value of the arc*. Thus the graph contains the information required by Algorithm T. If m oriented (non-identical) paths exist from u_l to u_i and the pth path is $(u_0^{(p)}, u_1^{(p)}, \ldots, u_{n_p}^{(p)})$, where $u_0^{(p)} = u_l$, $u_n^{(p)} = u_i$, then the coefficient $c_{i,l}$ of (8) can be expressed as

* If both operands (u_j and u_k) producing u_i are the same number then u_j and u_k are the same vertex, but the values ($d_{i,j}$ and $d_{i,k}$) of the two arcs leading from this vertex to u_i are in general different.

$$c_{i,l} = \begin{cases} \sum_{p=1}^{m} \prod_{q=1}^{n_p} d_{q,q-1}^{(p)}, & i \neq l, \\ 1, & i = l, \end{cases} \tag{26}$$

where $d_{q,q-1}^{(p)}$ denotes the value of the arc from $u_{q-1}^{(p)}$ to $u_q^{(p)}$. For example, in the graph illustrated in Figure 1 $c_{4,1} = d_{4,2}d_{2,1} + d_{4,3}d_{3,1} + d_{4,3}d_{3,2}d_{2,1}$. The graph is produced e.g. by the algorithm $u_2 \leftarrow u_0 + u_1$, $u_3 \leftarrow u_2/u_1$, $u_4 \leftarrow u_3 - u_2$, which can be programmed as $z := x + y$; $z := z - (z/y)$, where the values of the variables x and y are respectively u_0 and u_1 and the final value of z is u_4.

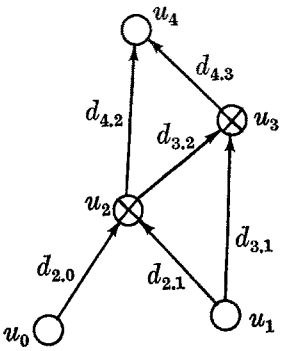

Figure 1. A computing process as a graph.

As noted in the preceding section, in many applications a weighted sum of the hth power of the $c_{i,p}$'s,

$$S_i = \sum_{p=1}^{i} s_p c_{i,p}^h, \tag{27}$$

where each weight s_p is a property connected with the number u_p, is required rather than the coefficients $c_{i,p}$ themselves. Such applications are the estimation of the expected value and variance of the accumulated rounding error, which can be estimated respectively with $h = 1$, $s_p = E(r_p)$ and $h = 2$, $s_p = D^2(r_p)$, $p = 1, \ldots, i$, where $E(r_p)$ is the expected value and $D^2(r_p)$ the variance of the local error r_p [1]. Naturally the question arises of whether it is necessary to save all the information to enable the reconstruction of the graph (as required by Algorithm T) when only the sum (27) is needed. This question is analyzed below.

At any given phase of a computing process each vertex u_i of the corresponding graph is either *active* or *inactive*, i.e. the number u_i may or may not appear as an operand in the remaining operations of the process. In practice the inactivity of u_i means that u_i is no longer saved as the contents of a cell in the main storage. When a vertex has been inactiv-

ated (by changing the contents of a cell) it can not be reactivated during the remainder of the process.

In particular, if $h=1$ in (27) then it follows from the linearity that for every vertex u_j which has no arcs leading to it each $d_{i,j}$ in the graph may be replaced by zero (and thus the arc from u_j to u_i eliminated), provided that the value s_i is simultaneously replaced by $s_i+d_{i,j}s_j$. Also each inactive vertex may be eliminated as soon as it is neither an initial nor a final vertex of any arc. Using these rules it is unnecessary to save the actual graph in order to compute S_i; S_i may instead be computed simultaneously with the computing process itself. The expression (27) then holds for the reduced graph, and thus $S_i=s_i$ when no more arcs leading to u_i are left.

When $h\neq 1$ the rules for eliminating parts of the graph are not as powerful. However, on the basis of the distributive law, three reduction rules exist such that (27) remains true for all values of h.

1. If several arcs lead from u_j to u_i, they can be replaced by a single arc whose value $d_{i,j}$ is the sum of the values of the original arcs.

2. If an inactive vertex u_j has exactly one arc leading from it, say to u_i, then this arc and the vertex u_j can be eliminated provided that s_i is simultaneously replaced by $s_i+s_jd_{i,j}{}^h$ and each arc leading to u_j, say from u_k, is replaced by an arc leading from u_k to u_i and having the value $d_{i,k}\leftarrow d_{i,j}d_{j,k}$.

3. If an inactive vertex has no arcs leading from it then it can be eliminated together with any arcs leading to it, since they can no more be used in (27).

The coefficients $c_{i,p}$ are unchanged by the reduction process and can be determined using (26). If Algorithm T is to be used, it must be modified in an obvious way, since there may now be several arcs leading to u_i instead of the two identified by $J[i]$ and $K[i]$ in Algorithm T.

Clearly the reduction rules given above do not lead to a complete reduction of the graph as in the case $h=1$. To illustrate the reduction achived three applications are given:

1. If the vertices u_2 and u_3 in Figure 1 are inactive (as they are e.g. after executing the statements $z:=x+y$; $z:=z-(z/y)$), then reduction gives the graph shown in Figure 2, in which

$$
\begin{aligned}
d'_{4.1} &= c_{4.1} = (d_{4.2}+d_{4.3}d_{3.2})d_{2.1}+d_{4.3}d_{3.1}\,,\\
d'_{4.0} &= c_{4.0} = (d_{4.2}+d_{4.3}d_{3.2})d_{2.0}\,,\\
s_4{}' &= s_4+s_3d_{4.3}^h+s_2(d_{4.2}+d_{4.3}d_{3.2})^h\,,\\
s_0{}' &= s_0 \quad \text{and} \quad s_2{}' = s_1\,.
\end{aligned}
$$

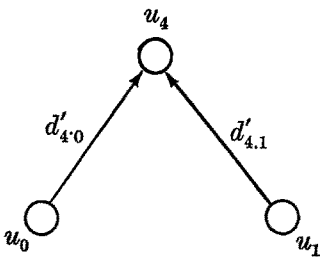

Figure 2. A reduced graph.

2. The original graph of Horner's scheme for computing the value of an nth degree polynomial contains $3n+3$ vertices (including those for the $n+2$ initial values) and $4n+1$ arcs. When the reduction rules are applied as soon as possible during the process, the maximum size of the graph is reduced to $n+4$ vertices and 3 arcs, i.e. only 2 vertices are needed in addition to the initial values, whatever the value of n. Thus the storage requirement remains proportional to the number of *variables*, and not to the number of operations as with the unreduced graph.

3. Suppose a linear system of m equations $Ax_i=f_i$, $i=1,\ldots,m$, where A is a square matrix of order n is solved using Gaussian elimination. Then the corresponding unreduced graph contains $2n^3/3+n^2/2-n/6+m\times(2n^2)$ vertices (including those for the n^2+mn initial values) and $4n^3/3-n^2-n/3+m\times(4n^2-2n)$ arcs. When the reduction rules are applied during the process, the maximum size of the graph obtained is $n^2+2+m\times(n^2+2n+1)$ vertices and $2n^3/3-n^2/2-n/6+m\times(2n^2-2)$ arcs. Thus the extra storage required can be roughly halved, but its magnitude is still proportional to the number of operations rather than to the number of variables.

7. Conclusions.

Both analytic and algorithmic methods of computing the coefficients of the Taylor expansion, especially the first and second order coefficients, have been considered. The algorithmic methods especially provide a convenient means of analyzing the effect of local errors upon the accumulated error. Analysis of the storage and time requirements shows that the methods are efficient enough to be used for testing the numerical stability of numerical algorithms, but their use would hardly be justified for continuous estimation of the behavior of rounding errors in routine computing processes, due principally to the large external storage requirement.

The method for computing the first order coefficients can easily be combined with estimates for the statistical behavior of the local rounding errors [3], thus providing an easy way of obtaining automatic estimates of the statistical behavior of accumulated rounding errors [1], [4]. Some experimental results obtained using this method are presented in [3] and [4].

Another use of the coefficients is for determining the most critical steps of a process, due to the information obtained regarding which local errors have the greatest effect upon the accumulated error of the resulting value.

Acknowledgements.

I am grateful to Professor Martti Tienari for his stimulating advice during the development of the present methods. Thanks are also due to Mr. Esko Ukkonen, M.Sc., with whom I have had valuable discussions. The language of this article was checked by Mr. Frank Beaver.

REFERENCES

1. P. Henrici, *Elements of numerical analysis*, Ch. 16, Wiley, New York, 1964.
2. S. Linnainmaa, *The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors*, (In Finnish), Master's Thesis, Department of Computer Science, University of Helsinki, Helsinki, Finland, 1970.
3. S. Linnainmaa, *Towards accurate statistical estimation of rounding errors in floating-point computations*, BIT 15 (1975), 165–173.
4. S. Linnainmaa, *A set of ALGOL procedures for analyzing the behavior of rounding errors in floating-point computations*, Report A-1975-2, Department of Computer Science, University of Helsinki, Helsinki, Finland.
5. M. Tienari, *A statistical model of roundoff errors for varying length floating-point arithmetic*, BIT 10 (1970), 355–365.
6. M. Tienari, *On some topological properties of numerical algorithms*, BIT 12 (1972), 409–433.

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF HELSINKI
TÖÖLÖNKATU 11
SF-00100 HELSINKI 10
FINLAND