GENERATION OF ERRORS IN DIGITAL COMPUTATION

ALSTON S. HOUSEHOLDER

Consider the problem of computing the numerical value $f(x)$ of some function f corresponding to a particular value of x. If x is a physical quantity whose value is known only from measurement, then one has available not x itself but some approximation x^*. If x is a mathematical constant, one may again be forced to replace x by some approximate value x^*. This is to be expected at least when x is irrational, and is often true when x is rational. For these or other reasons, though $f(x)$ is desired, one may have to accept some $f(x^*)$ instead.

Unless the function f is rational, one will generally be forced to represent f approximately by a truncated Taylor series, or an orthogonal series, or in some other fashion. In general, therefore, one does not strictly compute even $f(x^*)$, but rather some $f_a(x^*)$, where f_a represents a function which approximates f over some range containing x^*.

Suppose that f_a is defined by a finite sequence of elementary arithmetic operations, and that the computations are digital. In general the result of a division is not representable exactly by a terminating decimal. The same statement holds, of course, if one were using any fixed base. The result of multiplying two numbers, each expressible by n digits, will in general require $2n$ digits for its representation and further multiplications would increase the number of digits in proportion. Hence for either a product, or a quotient, one will in general replace the true result by an approximation obtained by truncating the sequence of digits and perhaps adjusting by some rule the last one retained. Hence one does not, in general, end up even with $f_a(x^*)$, but rather with some $f^*(x^*)$, in which true products and quotients are replaced by pseudo products and pseudo quotients, obtained according to rules that are determined in part by the nature of the facilities used for the computing.

Thus one starts to compute $f(x)$ and ends by computing some $f^*(x^*)$, thereby committing an error whose amount can be expressed as a sum of three independent components:

$$\begin{aligned} f(x) - f^*(x^*) = {} & [f(x) - f(x^*)] + [f(x^*) - f_a(x^*)] \\ & + [f_a(x^*) - f^*(x^*)]. \end{aligned} \tag{1}$$

An address delivered before the Spartanburg meeting of the Society on November 28, 1953 by invitation of the Committee to Select Hour Speakers for Southeastern Sectional Meetings; received by the editors December 4, 1953.

Of these three components, the first will be called the propagated error, the second the residual error, and the third the generated error.

If it is possible to assign limits to the error $x-x^*$, then by standard methods one can estimate the propagated error. The propagated error is completely independent of computing techniques and devices and can be reduced only by improving the approximation x^* to x.

The nature of the facilities available to the computor will guide him in his selection of the approximation f_a. So, too, will the requirements of accuracy: if a linear approximation will suffice, there is no point in computing the quadratic term. But once f_a has been determined, the residual error is also determined. It can be estimated by applying standard remainder formulas.

There remains, however, the generated error. It depends upon the facilities being used, and even upon quite subtle details of their application. This is the subject on which I wish to speak.

Unfortunately, there is no well developed general theory of generated error. For one thing the need did not become acute until the advent of automatic, high speed computing machinery. For another thing, the differences among machines now operating are such that each machine demands a theory all its own. Nevertheless, a few general principles can be stated, or at least illustrated and used as guides. This I shall attempt to do, first by describing a specific computing system, and then analyzing certain particular computational sequences.

In 1947, von Neumann and Goldstine gave an elaborate analysis of the errors generated in the course of inverting a matrix by Gaussian elimination. This is the first systematic treatment of generated error to appear in the literature. The computing system I shall presuppose is called the Oracle and is quite similar to theirs, but differs in a few respects. I shall assume that a register in the machine exhibits $\sigma+1$ binary digits, $\alpha_0, \alpha_1, \cdots, \alpha_\sigma$, and that the machine's arithmetic is designed with the assumption that these digits represent a number

$$a = -\alpha_0 + \alpha_1 \cdot 2^{-1} + \cdots + \alpha_\sigma \cdot 2^{-\sigma}.$$

Any number so represented will be called a digital number, so that digital numbers satisfy

$$-1 \leqq a < 1,$$

and are integral multiples of $2^{-\sigma}$. The binary digit α_0 is called the sign digit, the others the precision digits.

When multiplication is called for, the sign digit and the first σ

precision digits of the product will appear in one of the registers, and the remaining σ precision digits will appear in the other. Normally one retains only the most significant part, and this will be called the simple pseudo product. If the factors are a and b, the simple pseudo product will be designated $(ab)^*$. A special order will yield $ab+2^{-\sigma-1}$, whose most significant part is the rounded pseudo product $(ab)_r^*$. These pseudo products satisfy the relations

$$0 \leqq ab - (ab)^* < 2\epsilon, \qquad -\epsilon \leqq ab - (ab)_r^* < \epsilon, \tag{2}$$

where

$$\epsilon = 2^{-\sigma-1} \tag{3}$$

is the unit rounding error.

When multiplication is called for, and the pseudo product is to be sent to storage, then if $a=b=-1$, storage will receive -1. Hence in planning a computation one must be sure that this case cannot arise, or else provide remedial steps to be taken in case it does. For all other combinations of a and b, multiplication is legitimate.

For division a/b, it is required that $|a| < |b|$. Division can be performed with a digital, or with a of the form

$$a = a_1 + a_2 \cdot 2^{-\sigma},$$

where a_1 and a_2 are digital and $a_2 \geqq 0$. In either case, following the division one register will exhibit a truncated pseudo quotient $(a/b)^*$ which satisfies

$$\begin{aligned} 0 &\leqq a/b - (a/b)^* < 2\epsilon, & b > 0, \\ 0 &< a/b - (a/b)^* \leqq 2\epsilon, & b < 0. \end{aligned} \tag{4}$$

The other register will exhibit the remainder.

If n is a small positive integer the direct division of n into a digital number a can be formed by forming $2^{-\sigma}a/(2^{-\sigma}n)$. The somewhat more precise limits for the pseudo quotient are often useful:

$$\begin{aligned} 0 \leqq a/n - (a/n)^* &\leqq 2(n-1)\epsilon/n, \\ 2\epsilon/n \leqq a/(-n) - [a/(-n)]^* &\leqq 2\epsilon. \end{aligned} \tag{5}$$

Addition and subtraction propagate, but do not generate errors, unless the sum or difference falls outside the digital range. This would be considered a blunder and I shall assume it does not happen. Likewise I shall assume that any product $ab<1$, and that for any quotient $|a/b|<1$.

In general a mathematical formulation does not by any means define uniquely a computational procedure, and it is the computational procedure that determines the generated error. A particular ordered sequence of operations is called a routine, and the unambiguous prescription for such a sequence is a program. In planning the computation of a particular function, one wishes to program a routine that will minimize the maximum possible generated error, or that will at least hold the maximum possible generated error within tolerable bounds. This is not the only possible criterion, but it is the one that will be applied here.

Consider to begin with the very simple problem of computing $a/(bc)$. To avoid tedious multiplicity of cases, assume all factors positive, and assume, as usual, that any operation called for is legitimate.

The computation, then, seems eminently reasonable. We have a choice, however, of doing two divisions in sequence, or of doing first a multiplication and then a division. Machine-wise, as many steps are required one way as the other. Suppose the multiplication is done first. We have

$$\begin{aligned}\frac{a}{bc} - \left[\frac{a}{(bc)^*}\right]^* &= \frac{a}{bc} - \frac{a}{(bc)_r^*} + \frac{a}{(bc)_r^*} - \left[\frac{a}{(bc)_r^*}\right]^* \\ &= \frac{a}{bc(bc)_r^*}\left[(bc)_r^* - bc\right] + \frac{a}{(bc)_r^*} - \left[\frac{a}{(bc)_r^*}\right]^*.\end{aligned}$$

The second difference on the right is the difference between a true quotient and a pseudo quotient, and lies between 0 and 2ϵ. Consider, therefore, the first expression on the right.

Disregarding the trivial case $a=0$, which leads to a result that is strictly correct, the division could be legitimate with $(bc)_r^*$ as small as $2^{-\sigma+1}=4\epsilon$ provided $a=2\epsilon$. Since

$$-\epsilon \leqq bc - (bc)_r^* < \epsilon,$$

therefore ab can be as small as $4\epsilon-\epsilon=3\epsilon$. This is assuming that the limits have been attained and we have

$$a = 2\epsilon, \qquad (bc)_r^* = 4\epsilon, \qquad bc = 3\epsilon.$$

In this event, the first expression on the right becomes

$$\frac{2\epsilon}{3\epsilon\cdot 4\epsilon}\left[4\epsilon - 3\epsilon\right] = \frac{1}{6}\cdot$$

The computed quotient can be too small by 1/6, or 25% of the true quotient 2/3.

If bc and $(bc)_r^*$ differ in the other direction, the computed result can be too large by nearly 1/10, or again 25% of the true quotient which would, in this case, be 2/5.

This is, of course, an extreme, but perfectly possible case. Since the error is certainly intolerable, let us consider the other routine of doing successive divisions. We have now

$$\frac{a}{bc} - \left[\left(\frac{a}{b}\right)^* \Big/ c\right]^* = c^{-1}\left[\frac{a}{b} - \left(\frac{a}{b}\right)^*\right] + \left(\frac{a}{b}\right)^* \Big/ c - \left[\left(\frac{a}{b}\right)^* \Big/ c\right]^*.$$

Again it is the first expression that requires consideration. We now observe immediately that if b and c are different, then it will make a difference in our result whether we first divide by b, as indicated, or first divide by c instead. Suppose we have somehow arranged it so that $c \geqq b$, which gives the most favorable result. Then, since $cb > a$, necessarily

$$c > a^{1/2}.$$

Again neglecting the trivial case $a = 0$, we can assume $a \geqq 2\epsilon$ and therefore

$$c > (2\epsilon)^{1/2}.$$

Hence we come out with the result that

$$0 \leqq \frac{a}{bc} - \left[\left(\frac{a}{b}\right)^* \Big/ c\right]^* < (2\epsilon)^{1/2} + 2\epsilon. \tag{6}$$

This is considerably better, but even so only about half the figures are significant and we have assumed that c was known to be not less than b.

The extremely unfavorable result of the first routine occurs when bc is small. If bc is not so small the result might be more satisfactory. Assuming again that $c > b$, we shall say that the first routine is to be preferred over the second unless

$$\frac{a}{bc(bc)_r^*} \geqq \frac{1}{c},$$

or, since all factors are supposed positive, unless

$$b \leqq a/(bc)_r^*.$$

This is equivalent to saying that

$$b \leqq [a/(bc)_r^*]^*. \tag{7}$$

The second inequality implies the first since the pseudo quotient cannot exceed the true quotient. Conversely, the pseudo quotient and b are both digital. Hence if b exceeds the pseudo quotient, it must do so by at least as much as 2ϵ, whereas the true and pseudo quotients differ by less than 2ϵ.

It follows that if our pseudo quotient, obtained by following the first routine, is exceeded by both of the factors in the denominator, then we have made the best selection. If not, we should recompute, following the second routine. To show that the condition is not trivial, take $a=2^{-10}$, $bc=2^{-6}$. Then

$$a/(bc)_r^* = 2^{-4}$$

and condition (7) fails if $b=c=2^{-3}$, but is satisfied if $b=2^{-5}$, $c=2^{-1}$.

If one does not know a priori enough about the magnitudes of the quantities involved to select the preferred routine, or at least to know that one routine or the other, even if not the preferred one, will yield a satisfactory result, then it is quite possible to program both routines and the selection of the better of the two results. The machine will then follow the first method, test the criterion, and recompute by the second method if the criterion fails.

On the Oracle, the special feature permitting the direct use of a double precision dividend makes it possible to get a much better result and with less trouble. Suppose $b \leqq c$. The discrimination is easily programmed. The machine division of a by b yields the partial quotient and the remainder, so that one can store the partial quotient and continue the division to obtain the next σ digits of the quotient. Altogether, then, we have 2σ precision digits of the quotient a/b, which form a number $(a/b)'$ satisfying

$$0 \leqq a/b - (a/b)' < 4\epsilon^2.$$

If one divides $(a/b)'$ by c and denotes the result by $(a/bc)^*$, the error can be written

$$\frac{a}{bc} - \left(\frac{a}{bc}\right)^* = c^{-1}\left[\frac{a}{b} - \left(\frac{a}{b}\right)'\right] + c^{-1}\left(\frac{a}{b}\right)' - \left(\frac{a}{bc}\right)^*.$$

Since

$$bc > a, \qquad c \geqq b,$$

it follows that

$$c > a^{1/2}$$

and except for the trivial case $a = 0$,

$$a \geqq 2\epsilon.$$

Hence

$$c > (2\epsilon)^{1/2}.$$

Hence

$$0 \leqq \frac{a}{bc} - \left(\frac{a}{bc}\right)^* < 2\epsilon[1 + (2\epsilon)^{1/2}]. \tag{8}$$

The example I have just given is by no means artificial. Thus, iterated division is required in the formation of divided differences for purposes of interpolation, and the number of factors in the divisor is equal to the order of the divided difference. Let the divided differences be formed according to the iterative scheme

$$y_{01\cdots i} = (y_{12\cdots i} - y_{01\cdots i-1})/(x_i - x_0).$$

Mathematically, $y_{0\cdots i}$ is a symmetric function of its arguments. However, if the digital number $y^*_{01\cdots i}$ is to be formed and utilized in the computation of the divided difference of next higher order, then it is clear from the foregoing discussion that the optimal arrangement will be that for which

$$x_0 < x_1 < x_2 < \cdots$$

or that for which the inequalities are reversed. Unfortunately, the formation of a double precision dividend is more costly, since the dividend is the difference of two quantities, and the difference of double precision numbers requires a fair amount of programming.

It is usual to indicate the limits of error by estimating the absolute value of the difference between the computed and the desired results. However, this may lead to estimates that are unduly pessimistic. Thus, expressed in this form we would have to say that in division

$$|a/b - (a/b)^*| \leqq 2\epsilon,$$

which fails to exhibit the fact that the pseudo quotient cannot exceed the true quotient. More generally, suppose one carries out a sequence of operations, coming out at the end with some quantity f^*. We expect our error analysis to yield for us two numbers, which we may call δ_l and δ_u, which are such that the true value of f must satisfy

$$f^* - \delta_l \leqq f \leqq f^* + \delta_u. \tag{9}$$

If this can be done, then it is not at all important that δ_l and δ_u should be equal, or even that they should be both positive. If these are known then we can always, if we choose, replace f^* by another number that is as close as possible to the midpoint of the interval on which f is now known to lie. The important thing, therefore, in designing the program, is to make $\delta_u+\delta_l$ as small as possible.

This quantity $\delta_u+\delta_l$ I shall call the uncertainty (cf. Kuntzmann's "incertitude"), the interval from $f^*-\delta_l$ to $f^*+\delta_u$ will be the interval of uncertainty, and the two end points the limits of uncertainty. In general one will select a routine for which the uncertainty is as small as possible. Thereafter one can determine one or the other of the limits of uncertainty, and finally, perhaps, adjust the computed f^* to place it as close as possible to the midpoint of the interval of uncertainty.

Consider the evaluation of the first $n+1$ terms of a Maclaurin series. Presumably, up to some n at least, each additional term will increase the uncertainty but reduce the residual error. There will be some point beyond which further computation is unprofitable, either because the truncation error is already below tolerance, or because the decrease in the truncation error is overshadowed by the increase in the uncertainty. This point, moreover, will almost certainly vary with x. Hence one might be inclined to calculate and add successive terms until a term is reached whose value is less than some assigned quantity.

Each term in the series will have the form a_ix^i. If a_i is computed and multiplied by x^i, then one forms

$$(a_ix^i)^* = [a_i^*(x^i)^*]^*.$$

If we use roundoff multiplication, the error limits are symmetric. Suppose, for simplicity, that

$$|a_i - a_i^*| \leqq \alpha_i\epsilon.$$

If

$$|x^i - (x^i)^*| \leqq \xi_i\epsilon,$$

then

$$x^{i+1} - (x^{i+1})^* = x[x^i - (x^i)^*] + x(x^i)^* - (x^{i+1})^*.$$

Hence

$$\xi_{i+1} \leqq |x|\xi_i + 1.$$

But $\xi_1=0$, so that

$$\xi_i \leqq (1-|x|^{i-1})/(1-|x|) < i-1.$$

Also

$$a_ix^i - (a_ix^i)^* = (a_i - a_i^*)x^i + a_i^*[x^i - (x^i)^*] + a_i(x^i)^* - (a_ix^i)^*.$$

Hence, for $i \geqq 1$,

$$|a_ix^i - (a_ix^i)^*| \leqq \{\alpha_i |x|^i + |a_i^*| \xi_i + 1\}\epsilon.$$

The uncertainty in the term in x^i is twice the right member of this inequality, and the total uncertainty for $n+1$ terms is obtained by summing to $i = n$. If the a_i are of the order of unity, then for x of the order of unity the sum is of the order of $n^2\epsilon$.

If we are willing to fix n independently of x, or if we can select n in advance for any given x, then another procedure is possible. Let

$$y_n = a_n, \qquad y_i = xy_{i+1} + a_i.$$

Then by a backward induction, y_0 is the required true sum. Also

$$|y_n - y_n^*| \leqq \alpha_n\epsilon,$$

$$\begin{aligned} y_i - y_i^* &= xy_{i+1} - (xy_{i+1}^*)^* + (a_i - a_i^*) \\ &= x(y_{i+1} - y_{i+1}^*) + xy_{i+1}^* - (xy_{i+1}^*)_r^* + (a_i - a_i^*). \end{aligned}$$

Hence if

$$|y_i - y_i^*| \leqq \eta_i\epsilon,$$

then

$$\eta_i \leqq |x| \eta_{i+1} + 1 + \alpha_i,$$

$$\eta_0 \leqq \sum_0^n \alpha_i |x|^i + (1 - |x|^n)/(1 - |x|).$$

If $\alpha > \alpha_i$ for all i, then

$$\eta_0 < (n + 1)\alpha + n,$$

and if α is at worst a small integer the uncertainty is of the order of a small multiple of $n\epsilon$, instead of being of the order of $n^2\epsilon$.

Quite often the computation of the a_i can be incorporated into the induction so as to reduce the uncertainty still further. Consider the first $n+1$ terms of e^{-x}. Let

$$y_n = 1 + x/(-n), \qquad y_i = 1 + xy_{i+1}/(-i).$$

Then y_1 is the required sum. Then

$$y_i - y_i^* = \frac{x}{-i}(y_{i+1} - y_{i+1}^*) - \frac{1}{i}[xy_{i+1}^* - (xy_{i+1}^*)^*]$$

$$+\frac{(xy^*_{i+1})^*}{-i}-\left[\frac{(xy^*_{i+1})^*}{-i}\right]^*.$$

Let

$$-\eta_i'\epsilon \leqq y_i - y_i^* \leqq \eta_i''\epsilon.$$

Then, using simple multiplication,

$$\eta_i' = \frac{x}{i}\eta_{i+1}'', \qquad \eta_i'' = \frac{x}{i}\eta_i' + 2.$$

Hence if

$$\eta_i = \eta_i' + \eta_i''$$

so that $\eta_i\epsilon$ represents the uncertainty, then

$$\eta_i = 2 + x\eta_{i+1}/i.$$

Now $\eta_n \leqq 2$, and by a reverse induction

$$\frac{1}{2}\eta_1 \leqq 1 + \frac{x}{2} + \cdots + \frac{x^n}{n!} < e^x.$$

Hence, independently of n,

(10) $$\eta_n < 2e^x.$$

Thus $2\epsilon e^x$ is an absolute limit to the uncertainty. In general the inclusion of an additional term increases the uncertainty by $2\epsilon x^n/n!$, and decreases the residual error by an amount of the order of $x^n/n!$.

Continued fraction expansions sometimes have desirable properties, not the least being that they may converge where the power series does not. If the fraction is written in the form

$$F = b_0 + \frac{a_1}{b_1 +}\ \frac{a_2}{b_2 +}\ \cdots,$$

and if c_n represents either the numerator or the denominator of the nth convergent, then by the well known recursion

$$c_\nu = b_\nu c_{\nu-1} + a_\nu c_{\nu-2}.$$

The analysis is greatly simplified when the a's and b's are all positive. One may determine in advance that n for which

$$F_n = A_n/B_n$$

is sufficiently close to F. Then A_n^* and B_n^* are computed by applying the recursion to the A's and B's, and finally

$$F_n^* = (A_n^*/B_n^*)^*$$

is the computed value of F. Then

$$\begin{aligned} F_n - F_n^* = {} & B_n^{-1}\{(A_n - A_n^*) - (B_n - B_n^*)A_n^*/B_n^*\} \\ & + A_n^*/B_n^* - (A_n^*/B_n^*)^*. \end{aligned}$$

If $\alpha_n\epsilon$ and $\beta_n\epsilon$ represent the uncertainty in A_n and in B_n, respectively, then approximately the uncertainty in F_n is equal to

$$[B_n^{-1}(\alpha_n + F\beta_n) + 2]\epsilon.$$

From the recursion we have

$$\begin{aligned} c_\nu - c_\nu^* = {} & b_\nu(c_{\nu-1} - c_{\nu-1}^*) + a_\nu(c_{\nu-2} - c_{\nu-2}^*) \\ & + (b_\nu - b_\nu^*)c_{\nu-1}^* + (a_\nu - a_\nu^*)c_{\nu-2}^* \\ & + b_\nu^* c_{\nu-1}^* - (b_\nu^* c_{\nu-1}^*)^* + a_\nu^* c_{\nu-2}^* - (a_\nu^* c_{\nu-2}^*)^*. \end{aligned}$$

If $\gamma_\nu\epsilon$ represents the uncertainty (hence either $\alpha_\nu\epsilon$ or $\beta_\nu\epsilon$ as the case may be), then γ_ν satisfies a recursion of the form

$$\gamma_\nu = b_\nu\gamma_{\nu-1} + a_\nu\gamma_{\nu-2} + \delta_\nu,$$

where δ_ν represents a linear combination of the uncertainties in a_ν and b_ν, together with added terms arising from the products being formed. Hence the γ's satisfy a nonhomogeneous system whose matrix is identical with that of the system which determines the c's.

The properties of continuants are well known and I shall not discuss them here. However, it is clear that if the a's and b's are large, then the uncertainty will build up rapidly. In fact, it builds up much more rapidly in proportion than does c_n, because of the presence of the nonhomogeneous term in each equation. Moreover, we may be faced with a difficult scaling problem. On the other hand, if the a's and b's are too small, B_n may be quite small, and the factor B_n^{-1} in the final uncertainty will cause trouble. The best plan seems to be to make every $a_\nu = 1$ if the b_ν are then small, or else every $b_\nu = 1$ if the a_ν are then small. This does not necessarily eliminate the scaling problem, but if both $a_\nu \leqq 1$ and $b_\nu \leqq 1$, then scaling can be effected by introducing factors 2^{-1} with sufficient frequency.

In illustration of a computation of a different sort, consider the extraction of a square root by Newton's method. For solving $x^2 - a = 0$, it is convenient to write the iteration in the form

$$x_{i+1} = x_i - \Delta^*(x_i), \qquad \Delta(x) = (x - a/x)/2. \tag{11}$$

I have yet to define Δ^*. If $x_0 = 1$, it is easily shown that the mathematical sequence, defined with Δ instead of Δ^*, converges monotonically. One may expect, therefore, that the digital sequence will be monotonic at least up to some x_i, after which either Δ^* vanishes, or else the x_i become periodic in some manner.

There are several possible routines for computing a Δ^*, all differing slightly from one another, and all giving slightly different results. It turns out that if one programs the computation in the form

$$-\Delta^* = \{[-x - (-a/x)^*]/2\}^*,$$

then one can show that the sequence $-\Delta^*(x_i)$ increases monotonically to zero, and that when $\Delta^*(x) = 0$, then

$$-\epsilon + (a + \epsilon^2)^{1/2} \leqq x < \epsilon + (a + \epsilon^2)^{1/2}; \tag{12}$$

furthermore, for the same x, if $x \leqq 2^{-1}$, then

$$(x^2)_r^* = a,$$

and if $x > 2^{-1}$, then

$$|(x^2)_r^* - a| \leqq 2\epsilon.$$

One could not, of course, expect that the pseudo square of the pseudo square root would always equal the number itself, since numbers below, say, $\epsilon^{1/2}$ could not be pseudo square roots by any reasonable approximation. In other words, there are more digital numbers than pseudo square roots. The uncertainty in the square root obtained in this manner is exactly 2ϵ, which is the smallest possible value. Since

$$0 < (a + \epsilon^2)^{1/2} - a^{1/2} \leqq \epsilon,$$

with equality holding only when $a = 0$, therefore

$$-\epsilon < x - a^{1/2} < 2\epsilon.$$

Hence if $a^{1/2}$ is digital the routine always yields the strictly correct result.

In the case of cube roots, the best method devised so far leaves an uncertainty greater than 2ϵ by roughly $\epsilon^{3/2}$.

More generally, consider the problem of solving an equation

$$f(x) = 0$$

for a particular root α that is simple and has been isolated. Suppose, by some means, one has found some approximation x' such that

$$(13) \qquad \delta_l(x') \geqq f^*(x') \geqq -\delta_u(x').$$

For definiteness suppose f is increasing, and suppose, moreover, that an $m>0$ is known such that $f' \geqq m$ throughout an interval containing both α and x'. Then

$$(14) \qquad x' - (f^* + \delta_u)/m \leqq \alpha \leqq x' - (f^* - \delta_l)/m.$$

Since (14) holds for any x' satisfying (13), one might replace x' by $x' \pm 2\nu\epsilon$ and improve the limits slightly.

There are, of course, many ways of writing an equation $f(x)=0$ satisfied by α, and for each f, many possible possible routines for evaluating it. The limits (14) are optimal for a given f and a given routine for evaluating it, but will, in general, be different for different routines.

If one is to employ Newton's method, forming the sequence

$$x_{i+1} = x_i - \Delta^*(x_i),$$

where

$$\Delta(x) = f(x)/f'(x),$$

then the equation $\Delta=0$ is equivalent to $f=0$, and m is close to unity. If Δ is computed as a quotient, then one can write

$$\Delta - \Delta^* = [f - f^* - \Delta(f' - f'^*)]/f'^* + [(f^*/f'^*) - (f^*/f'^*)^*].$$

Near $x=\alpha$, Δ should be a small multiple of ϵ, and therefore the second term in the first bracket should be negligible. Hence approximately, in the vicinity of the root,

$$(15) \qquad \Delta - \Delta^* = (f - f^*)/f'^* + [(f^*/f'^*) - (f^*/f'^*)^*].$$

The first term on the right represents again the uncertainty in f divided by the derivative, but this uncertainty is increased by the fixed amount 2ϵ resulting from the division.

Of the many iterations of higher order that have been proposed, it seems unlikely that they can prove useful for this type of computation except possibly in very special cases. Even the simple-minded scheme of successive bisection requires at most σ evaluations of f, σ being, for most machines, approximately 40. Newton's method requires the evaluation of f' along with f, but should converge in fewer steps. An iteration of higher order will generally require a more complicated program for each step, with a corresponding increase in the uncertainty. What little can be gained by fewer steps is apt to be more than lost in the program.

My remarks here have not been directed to the experienced computors, by whom most of my conclusions are very likely well understood already. But however well known these matters may be, they do not seem to have been written down. The systematic study of generated error may be said to have begun with the paper by von Neumann and Goldstine [9] to which reference was made above. More recently Lotkin and Remage [7; 8] have analyzed the generation of error in matrix inversion by a different method. Goldstine (unpublished) has discussed the generation of error in the determination of the proper values of Hermitian matrices using an iterative scheme, and Givens [2], in a recent report, has analyzed the application of a certain direct method to the same problem. Other studies are made in memoranda and reports from various research organizations where computing machines are in operation.

My purpose here has been to indicate how the techniques can be applied to a variety of situations, and to point up the need for a systematic attack so as to transform the art of computing into a science of computing.

References

1. Paul S. Dwyer, *Errors of matrix computations, simultaneous linear equations and the determination of eigenvalues*, U. S. Dept. of Commerce, Applied Mathematics series, vol. 29 (1953) pp. 49–58.

2. J. W. Givens, *Numerical computation of the characteristic values of a real symmetric matrix*, ORNL Report 1574, 1953.

3. Saul Gorn, *On the study of computational errors*, Ballistic Research Laboratories, Report No. 816, 1952.

4. A. S. Householder, *Errors in iterative solutions of linear systems*, Proceedings of the Association for Computing Machinery, Meeting at Toronto, Ont. Sept. 8–10, 1952, pp. 30–33.

5. ———, *Principles of numerical analysis*, McGraw-Hill, 1953.

6. J. Kuntzmann, *Notions de grille et de tube*, Ann. Inst. Fourier vol. 2 (1950) pp. 197–205.

7. Max Lotkin and R. Remage, *Matrix inversion by partitioning*, Proceedings of the Association for Computing Machinery, Meeting at Toronto, Ont. Sept. 8–10, 1952, pp. 36–41.

8. ———, *Scaling and error analysis for matrix inversion by partitioning*, Ann. Math. Statist. vol. 24 (1953) pp. 428–439.

9. John von Neumann and H. H. Goldstine, *Numerical inversion of matrices of high order*, Bull. Amer. Math. Soc. vol. 53 (1947) pp. 1021–1099.

10. A. M. Turing, *Rounding off errors in matrix processes*, Quart. J. Mech. Appl. Math. vol. 1 (1948) pp. 287–308.

Oak Ridge National Laboratory