A Note on an Iterative Method for Root Extraction

by J. C. Gower

Summary: A double iterative method for evaluating $y/x^{1/n}$ is derived and it is shown that if $-1 \leqslant y^n < x < 1$ then it can be arranged that all terms occurring in the iteration are also within this range. The rate of convergence is then discussed and some special cases are mentioned.

INTRODUCTION

A method commonly used on electronic computers as a basis for a square root subroutine is the double iterative procedure defined as follows:

$$\left.\begin{aligned} a_{k+1} &= a_k(1 - \tfrac{1}{2}c_k) \\ c_{k+1} &= \tfrac{1}{4}c_k^2(c_k - 3). \end{aligned}\right\} \qquad (1)$$

If initially $a_0 = y$ and $c_0 = x - 1$, then it can be shown that $c_k \to 0$ and simultaneously $a_k \to y/\sqrt{x}$. By setting $y = x$ this gives the result $a_k \to \sqrt{x}$.

The method seems to have originated with the EDSAC group at Cambridge (Wilkes, Wheeler and Gill, 1951), and was adopted on several of the earlier computers; at Manchester and Rothamsted for instance. There are three reasons why the scheme was particularly suitable for machines built at that time. Firstly, the method only requires a few cycles of iteration (see Table 1). Secondly, no division is necessary, a great saving on those machines which require a special subroutine for this operation. Lastly, if $y^2 < x < 1$ then all subsequent values of a_k, c_k are fractional so that there is no danger of overflow in the accumulator. With computers with a built-in division order and facilities for working in floating-point arithmetic, the last two of these advantages have become of less importance. For such machines the well-known iterative procedure $a_{k+1} = \frac{1}{2}(a_k + x/a_k)$ is adequate for evaluating $\sqrt{x}$ and, in fact, has the advantage that round-off errors cannot accumulate.

The object of this note is to throw more light on the mechanism of this double iterative procedure and its rate of convergence, and to show how it can be generalized to evaluate $y/x^{1/n}$. The results are of general interest and may be of value even when working on a floating-point machine.

DERIVATION OF THE GENERAL FORMULAE

Consider the double iterative process:

$$a_{k+1} = a_k(1 + \alpha c_k) \qquad (2)$$

$$(y^n/x)(1 + c_{k+1}) = a_{k+1}^n \ . \qquad (3)$$

Evidently if $c_k \to 0$ as k increases then $a_k \to y/x^{1/n}$.

Equations (2) and (3) imply that:

$$(1 + c_{k+1}) = (1 + c_k)(1 + \alpha c_k)^n. \qquad (4)$$

The constant α is to be chosen so that $c_k \to 0$ as k increases, and so that this convergence is as rapid as possible. The best that can be done with (4) is to ensure a second-order process in c_k by defining α such that the term in c_k vanishes.* This gives $\alpha = -1/n$.

(2) and (4) now become:

$$a_{k+1} = a_k(1 - c_k/n) \qquad (5)$$

$$c_{k+1} = (1 + c_k)(1 - c_k/n)^n - 1. \qquad (6)$$

If $c_k \to 0$ then by (3), $a_k \to y/x^{1/n}$.

As initial conditions choose $a_0 = y$ and $c_0 = x - 1$, values which are consistent with (3). For the particular case $n = 2$, (5) and (6) give rise to the formulae (1). It is easy to show by an elementary, though rather lengthy, study of (6) that if c_k is a negative fraction

$$-1 \leqslant -c^2 \leqslant c_{k+1} \leqslant (1 + c_k)\exp(-c_k) - 1 < 0. \qquad (7)$$

If $0 < x < 1$ then $-1 < c_0 < 0$ and it immediately follows from (7) that all values of c_k are negative fractions which converge to zero as k increases. It further follows from (3) that $|a_k|^n < |y^n/x|$ and therefore a_k is fractional for all values of k provided that the final result $y/x^{1/n}$ is itself fractional.

The above has shown that the iterative procedure defined by (5) and (6), together with the initial con-

TABLE 1

THE NUMBER OF ITERATIONS REQUIRED FOR c_k TO BECOME LESS THAN 2^{-31}, FOR VARIOUS VALUES OF x

Number of iterations required	*Minimum value of x*
1	0·99998
2	0·99609
3	0·9375
4	0·75
5	0·5
6	0·2929
7	0·1591
8	0·0830
9	0·0424
10	0·0214

* To obtain a third-order process we may introduce a quadratic term βc_k^2 into (2), which gives rise to:

$$(1 + c_{k+1}) = (1 + c_k)(1 + \alpha c_k + \beta c_k^2)^n.$$

Equating the coefficients of c_k and c_k^2 to zero yields values of α and β which give the desired third-order process. Higher-order processes may be obtained in a similar way but the formulae so obtained become complicated unless $n = 1$.

ditions $a_0 = y$, $c_0 = x - 1$, is suitable for the evaluation of $y/x^{1/n}$ on an electronic computer working in fixed-point arithmetic, provided that the signs of x and y are first adjusted to ensure that x is positive.

The asymptotic form of (6) is:

$$c_{k+1} = (1 + c_k) \exp(-c_k) - 1. \qquad (8)$$

This formula is of value in discussing the rate of convergence of c_k and in theory gives a method for evaluating $y/x^{1/n}$ for large values of n. The accuracy of such a method would depend on the error in the equivalence of $\exp(-c_0)$ and $(1 - c_0/n)^n$, and is unlikely to be of much use in practice.

The above derivation does not require n to be integral. However, for non-integral values the evaluation of (6) would itself require some form of root extraction unless n is sufficiently large for formula (8) to be used.

CONVERGENCE

Formula (7) gives narrow bounds to the convergence process for c_k and it is a remarkable fact that this convergence is almost independent of n. This may be seen by drawing the curves $c_{k+1} = -c_k^2$ $(n = 1)$ and that of formula (8) $(n = \infty)$ which correspond to the slowest and fastest rates of convergence respectively. These two curves diverge from each other only slightly and enclose the curves corresponding to all the other values of n. In view of this an adequate idea of the behaviour of the convergence for all values of n can be obtained by discussing the slowest case (corresponding to $n = 1$, or division). Table 1 gives some figures for the number of iterations required for c_k to become less than 2^{-31} for various values of x. It will be noticed that convergence becomes slower as x decreases, and is actually infinitely slow when $c_0 = -1$ (i.e. when x is zero).

To make x as large as possible and so reduce the number of iterations, it should be arranged that $\frac{1}{2}^n \leqslant x < 1$. This can be done by multiplying x by the largest permissible power of 2^n, y being multiplied by the same power of 2. Apart from improving convergence this also improves accuracy by preserving more digits throughout the computation. Table 2 gives the average number of iterations required for various values of n, assuming that all x in the range $\frac{1}{2}^n \leqslant x < 1$ are equally likely and that the convergence is governed

TABLE 2

AVERAGE NUMBER OF ITERATIONS REQUIRED FOR c_k TO BECOME LESS THAN 2^{-31} IF x IS SCALED SO THAT $\frac{1}{2}^n \leqslant x < 1$.

n	*Average number of iterations*
1	4·37
2	5·05
3	5·29
4	5·49
5	5·60
6	5·68
7	5·73
8	5·76
9	5·78
10	5·78
20	5·80

sufficiently closely by $c_{k+1} = -c_k^2$. The actual performance will be slightly better than this.

SPECIAL CASES

(i) $n = 1$

$$\left.\begin{aligned} a_{k+1} &= a_k(1 - c_k) \\ c_{k+1} &= -c_k^2. \end{aligned}\right\}$$

This is a perfectly valid basis for a division subroutine.

(ii) $n = 2$ The formulae (1) are arrived at, giving a subroutine for $y/\sqrt{x}$ or $\sqrt{x}$ as used on many computers.

(iii) $n = 3$

$$\left.\begin{aligned} a_{k+1} &= a_k(1 - \tfrac{1}{3}c_k) \\ c_{k+1} &= \tfrac{1}{27}c_k^2(18 - 8c_k + c_k^2). \end{aligned}\right\}$$

This will evaluate $y/x^{\frac{1}{3}}$ or $y^{\frac{1}{3}}$ if we set $c_0 = y^2 - 1$.

A theoretical investigation of the effect of rounding errors is difficult, but the method has been in use for values of n up to $n = 3$ and has been found to be satisfactory. More experience is needed of the behaviour of the process for higher values of n but, even if rounding errors do become serious, the method can be used to give a very good first approximation for some trial and error method where the result is built up digit by digit.

REFERENCE

(1) WILKES, M. V., WHEELER, D. J., and GILL S. (1951). *The Preparation of Programs for an Electronic Digital Computer.* Cambridge, Mass.: Addison-Wesley Press.