W. L. Miranker

Parallel Methods for Approximating the Root of a Function

Abstract: We present a class of methods for approximating the root of a function. The methods are designed for execution on a parallel processor and when they are so executed, the speed of the approximation process is increased. The increase in speed is estimated analytically by computations of the order of convergence of the various methods presented.

1. Introduction

In this paper we consider the problem of speeding up the process of approximating a root of a function by iterative methods which use function values only. The increase in speed is achieved through the use of a parallel computer, i.e., a computer with a number of arithmetic processors capable of simultaneous and independent operation. We do not discuss all of the problems that pertain to the organization of parallel methods on a parallel machine. Rather, we show how the logical independence of the subparts of a class of algorithms may be exploited to increase speed of computation.

The method presented here produces an iterative sequence of r-vectors, each of whose components is supposed to converge to the root in question. The actual computation process parcels out the computing among, say, r processors in the computer. The j-th processor makes a function evaluation at the j-th component of the n-th iterate. Then the j-th processor computes the j-th component of the $n+1$-st iterate as an appropriate function of the function values just obtained by all of the processors. The process continues until a convergence test is satisfied. Many variations of this basic procedure are possible.

In Section 2 the class of methods is described. In Section 3 the order of convergence of the methods is obtained. For example, the standard method of Regula Falsi has order of convergence $(1+\sqrt{5})/2$, but the simplest two processor method presented here has order 2.2, approximately. In Section 4 an asymptotic calculation is made to compare the gains in speed of some of the methods described. In addition, a numerical example and suggestions for other parallel methods are given.

The author is a staff member at the IBM Watson Research Center, Yorktown Heights, New York. He is Visiting Professor, Institute of Mathematics, The Hebrew University of Jerusalem, until June 1969.

2. Description of method and order of convergence

Let $f(x)$ be the function whose root z is sought and let $\mathbf{x}_n = (x_n^1, \cdots, x_n^r)$ be an r-vector, each of whose components is considered as an approximation to z. The next approximation $\mathbf{x}_{n+1}$, is determined as follows: choose an integer $m \geq 2$. The integer m is independent of n. Determine r Lagrange interpolation polynomials, $L_{m+j}(x)$ of degrees $m+j-2, j=1,2,\cdots,r$. For each j, $j=1,\cdots,r$, $L_{m+j}(x)$ interpolates the points

$$(x_\alpha^\beta, f(x_\alpha^\beta)) \equiv (x_\alpha^\beta, f_\alpha^\beta) \tag{1}$$

with

$$\beta \equiv 1 + r - r\left[\frac{\gamma}{r} - \left(\frac{\gamma}{r}\right)\right] - r\,\delta\left[0, \frac{\gamma}{r} - \left(\frac{\gamma}{r}\right)\right]$$

$$\alpha = n - [(\gamma - 1)/r)]$$

$$\gamma = 1, 2, \cdots, m + j - 1.$$

Here the square brackets denote the integer part and the δ is the Kronecker delta of its two arguments. Let x_{n+1}^j, $j=1,\cdots,r$ be a root of $L_{m+j}(x)$. We denote this prescription for determining $\mathbf{x}_{n+1}$ by the function F_m of $2+[(m-1)/r]$ vector variables, viz:

$$\mathbf{x}_{n+1} = F_m(\mathbf{x}_n, \mathbf{x}_{n-1}, \cdots, \mathbf{x}_{n-1-[(m-1)/r]}). \tag{2}$$

Figure 1 illustrates the method for $r=2$, $m=2$.

The associated vector $\mathbf{F}_2 = F_2(\mathbf{x}_n, \mathbf{x}_{n-1})$ is given by

$$\mathbf{F}_2 = \left(\frac{x_n^1 f_n^2 - x_n^2 f_n^1}{f_n^2 - f_n^1}, \frac{f_n^1 f_n^2 x_{n-1}^2}{(f_{n-1}^2 - f_n^1)(f_{n-1}^2 - f_n^2)} + \frac{f_{n-1}^2 f_n^2 x_n^1}{(f_n^1 - f_{n-1}^2)(f_n^1 - f_n^2)} + \frac{f_{n-1}^2 f_n^1 x_n^2}{(f_n^2 - f_{n-1}^2)(f_n^2 - f_n^1)}\right) \tag{3}$$

We see that two new approximations x_{n+1}^1 and x_{n+1}^2 to the root are obtained from three current approximations

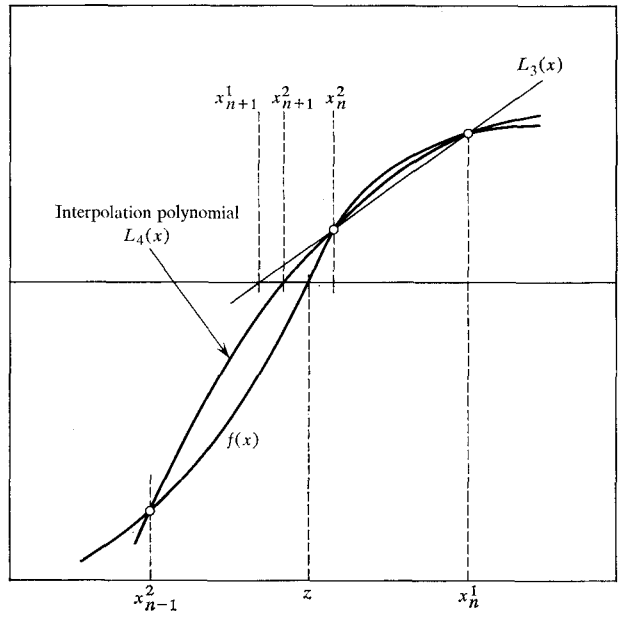

Figure 1 A step in the case $r = m = 2$.

x^2_{n-1}, x^1_n and x^2_n. Geometrically, we see that x^1_{n+1} is determined from the chord associated with x^1_n and x^2_n while x^2_{n+1} is determined from the parabola associated with x^2_{n-1}, x^1_n and x^2_n.

Heuristic description of calculation: The parameter r may be chosen in many ways. To fix ideas, let it be the number of processors available for the root calculation. At a given stage in the algorithm, we have produced a finite sequence of approximations to the root (of which only the last $m + r - 1$ need be kept in memory). The first processor uses the last m approximations and computes a new approximation by means of the interpolation polynomial L_{m+1} as described above. Simultaneously, the second processor uses the last $m + 1$ approximations to compute a new approximation by means of L_{m+2}. In this manner, the r processors produce r new approximations to the root which are appended to the full list of approximations. The first processor appends first, the second processor appends the second, etc. (The r oldest members of the list retained in memory may then be erased.) These iterations are to continue until a convergence test is passed.

Point of view: There are many problems of a technical nature concerning the class of algorithms just described. For example: a denominator in (3) might vanish so that the algorithm is not defined, the algorithm might not converge, the iterates might not be real even for real functions $f(x)$, etc. These problems are not a novel characteristic of the methods presented here, but are intrinsic difficulties of iterative methods for finding roots. See, for example, the text of J. Traub[2] where these questions are treated by a variety of means. Our point of view is to study the special feature of parallelism contained in our algorithms. We thus assume that the algorithms are well defined, that they converge, and if necessary, that the iterates are real. We are interested only in determining by how much we increase the speed of computation when we use the parallel mode of computation. The speed of computation is characterized by means of the order of convergence of the sequence of iterates. The order of convergence is assumed to exist, and in the next section we study the orders of our algorithms.

The question of determining iterates which are roots of high order polynomials may be raised as a serious limitation on all higher order extrapolatory methods including the ones presented here. As is well known, the device of inverse interpolation is a practical way of dealing with this problem. This technique replaces polynomial root finding by polynomial evaluation.

3. Order of convergence

Let ρ_n be the Euclidean distance between the r-vectors $\mathbf{x}_n(x^1_n, \cdots, x^r_n)$ and $\mathbf{z}(z, \cdots, z)$, where the latter denotes r copies of a root z of $f(x)$, i.e.,

$$\rho_n \equiv \rho(\mathbf{x}_n \mathbf{z}) \equiv \left[\sum_{j=1}^{r} (z^j_n - z)^2 \right]^{\frac{1}{2}}. \tag{3.1}$$

Let λ and C be positive constants such that

$$\rho_n \leq C^{\lambda^n}. \tag{3.2}$$

The largest λ for which (3.2) holds is the order of convergence of the sequence of iterates $\{\mathbf{x}_n\}$. (This notion is useful only if $C < 1$ and $\lambda > 1$.)

In order to apply higher-order interpolation methods and to derive orders of convergence, it is usual to assume that the functions being interpolated are sufficiently differentiable. We henceforth assume that our functions have as many derivatives as are necessary for our arguments. We also assume that the iterative scheme is convergent, and so considerations may be viewed as confined to a fixed compact interval about the root z of $f(x)$. The following theorem characterizes the order of convergence of our methods.

Theorem 1: The order of convergence of the sequence $\{x_n\}$ corresponding to the method with parameters (r, m) is the largest positive root of the polynomial

$$\det \left| A\lambda^{d+1} - \sum_{k=0}^{d} B_k \lambda^{d-k} \right| = 0. \tag{3.3}$$

Here $d = [(m + r - 1)/r]$ and $\mathbf{A}$ and the $\mathbf{B}_k$ are $r \times r$ matrices. $\mathbf{A} = (a_{ij})$ and the $\mathbf{B}_k = (b^k_{ij})$ are defined as follows:

$$a_{ij} = \begin{cases} 1 & i = j \\ -1 & i = j + 1 \\ 0 & \text{otherwise,} \end{cases} \tag{3.4}$$

$$b_{ij}^{k} = \begin{cases} 1 & i = r \qquad k = 0, \cdots, d - 2 \\ 0 & \text{otherwise,} \end{cases} \tag{3.5}$$

$$b_{ij}^{d-1} = \begin{cases} -1 & j = i + m + (1 - d)r \\ 1 & i = r \\ 0 & \text{otherwise,} \end{cases} \tag{3.6}$$

$$b_{ij}^{d} = \begin{cases} -1 & i = j + m - rd, \quad i \neq r \\ 1 & i = r, \quad j = m + (1 - d)r - 1 \\ 0 & \text{otherwise.} \end{cases} \tag{3.7}$$

Remark: In the known nonparallel cases corresponding to $r = 1$ and $d = m$ the matrix $\mathbf{B}_d$ becomes the scalar zero. The polynomial (3.3) becomes λ times the usual polynomial defining the order of convergence in this case. The null root is extraneous.

Examples of the polynomial (3.3) and its positive root are:

For $r = m = 2$,

$$\lambda(\lambda^3 - 2\lambda^2 - 1) \tag{3.8}$$

with positive root $\lambda = 2.19$.

For $r = 3, m = 2$,

$$\lambda^2(\lambda^4 - 3\lambda^3 + \lambda^2 - 1) \tag{3.9}$$

with positive root $\lambda = 2.57$.

For $r = 3, m = 3$,

$$\lambda(\lambda^5 - 3\lambda^4 - 3\lambda^2 - 1) \tag{3.10}$$

with positive root $\lambda = 3.59$.

The following theorem gives lower bounds for the orders of convergence. In the case $r \geq m$ an analytic expression for the order of convergence is readily obtainable from the theorem. This expression will be used in Section 4 to characterize the gain in speed per processor used.

Theorem 2: A lower bound for the order of convergence of the sequence $\{x_n\}$ corresponding to the method with parameters (r, m) is the largest positive root of

$$\lambda^2 - \frac{m}{2}\lambda - \frac{mr}{2}, \quad \text{if} \quad r \geq m, \tag{3.11}$$

or

$$\lambda^{2[m/r]} - \sum_{k=0}^{[m/r-1]} \left(\frac{r}{2} \lambda^{2[m/r]-k-1} + \frac{r^2}{2} \sum_{\beta=1}^{[m/r]} \lambda^{2[m/r]-k-\beta-1} \right), \quad \text{if} \quad r < m. \tag{3.12}$$

The roots of (3.11) corresponding to the three examples, (3.8)–(3.10), are respectively $\lambda = 2., 2.3, 2.38$. We see that the bounds provided by Theorem 2 are poor. However, the order of magnitude results obtained from Theorem 2 for use in Section 4 are probably as good as can be obtained.

As a preliminary to proving these theorems, we recall two known properties of Lagrange interpolation polynomials (see Ref. 1, where these properties are derived). Let e_n^i be the scalar, $e_n^i = |z - x_n^i|$. The first property is

$$e_n^i < C\,|f(z) - L_{m+i}(z)|. \tag{3.13}$$

Here, C is some constant and L_{m+i} is the Lagrange interpolation polynomial of degree $m + j - 2$ associated with the scalar iterate x_n^i.

$$|f(z) - L_{m+i}(z)| < C_{m+i} \prod_{k=r-m-j+2}^{r} e_n^k. \tag{3.14}$$

Eq. (3.14) is the standard error estimate for Lagrange interpolation polynomials. In (3.14) the superscript of e_n^k may take on nonpositive values. The meaning of this is clear and is given by $e_n^0 = e_{n-1}^r$, $e_n^{-1} = e_{n-1}^{r-1}$, $\cdots$, $e_n^{-r} = e_{n-2}^r$, $\cdots$. This convention will be used in what follows. In (3.13) and (3.14), we have not denoted the dependence of the Lagrange polynomials on the given points $(x_\alpha^\beta, f_\alpha^\beta)$ which they interpolate. Since the dependence may be identified from the context, we have, for clarity, eliminated the expression of this dependence.

Proof of Theorem 1: To prove Theorem 1, we combine (3.13) and (3.14) to get

$$e_{n+1}^r = CC_{m+r} e_n^r \cdots e_n^1 e_{n-1}^r \cdots e_{n-d}^{m+(1-d)r-1} \tag{3.15}$$

From this we readily derive the following relations:

$$\begin{aligned} e_{n+1}^{r-1} &= C \frac{C_{m+r-1}}{C_{m+r}} \frac{e_{n+1}^r}{e_{n-d}^{m+(1-d)r-1}} \\ &\cdots \\ e_{n+1}^{r-j} &= C \frac{C_{m+r-j}}{C_{m+r-j+1}} \frac{e_{n+1}^{n-j+1}}{e_{n-d}^{m+(1-d)r-j}} \\ &\cdots \\ e_{n+1}^{1} &= C \frac{C_1}{C_2} \frac{e_{n+2}^2}{e_{n-[(m+1)/r]}^{m+1-r[(m+1)/r]}} \end{aligned} \tag{3.16}$$

Next we take the logarithm of the equations (3.15) and (3.16). This gives a system of difference equations whose characteristic polynomial is (3.3). This completes the proof of Theorem 1.

Proof of Theorem 2: To prove Theorem 2, we combine (3.1) and (3.13) to get

$$\rho_{n+1} \leq C \left[\sum_{j=1}^{r} (f(z) - L_{m+i}(z))^2 \right]^{\frac{1}{2}}. \tag{3.17}$$

Then we combine (3.14) and (3.17) to get

$$\rho_{n+1} \leq C\left[\sum_{j=1}^{r}\left(C_{m+j}\prod_{k=r-m-j+2}^{r} e_n^k\right)^2\right]^{\frac{1}{2}}. \tag{3.18}$$

From (3.18) we may derive the difference inequalities

$$\rho_{n+1} \leq C\rho_n^{m/2}\rho_{n-1}^{mr/2} \qquad r \geq m \tag{3.19}$$

$$\rho_{n+1} \leq C\prod_{k=0}^{[m/r]-1}\left(\rho_{n-k}^{r/2}\prod_{\beta=1}^{[m/r]}\rho_{n-k-\beta}^{r^2/2}\right) \qquad r < m. \tag{3.20}$$

These derivations are given below. From (3.19) and (3.20), trial solutions for ρ_n of the form (3.2) give the characteristic equations (3.8) and (3.9) for λ.

In what follows, C will stand for a floating constant which changes its value as the arguments proceed.

Derivation of (3.19) *from* (3.18). We write (3.18) as

$$\rho_{n+1} \leq C\left[\sum_{j=1}^{r} C_{m+j}^2\left(\prod_{k=n-m-j+2}^{r} e_n^k\right)\cdot\left(\prod_{\substack{i=r-m-j+2\\ r-m-j+2}}^{r}\right)\left(\prod_{\substack{i=n-m-j+2\\ r-m-j+2\leq 0}}^{r} e_n^i\right)\right]^{\frac{1}{2}} \tag{3.21}$$

Since the iteration process converges, we may absorb the last product here into C. In the next to the last product, we estimate e_n^i by combining (3.13) and (3.14). Then, (3.21) becomes

$$\rho_{n+1} \leq C\left[\sum_{j=1}^{r} C_{m+j}^2\left(\prod_{k=r-m-j+2}^{r} e_n^k\right)\cdot\prod_{\substack{i=n-m-j+2\\ r-m-j+2>0}}^{r}\left(C_{m+i}\prod_{l=r-m-i+2}^{r} e_{n-1}^l\right)\right]^{\frac{1}{2}}, \tag{3.22}$$

where the constant appearing in (3.14) raised to an appropriate power has been absorbed into C. Next we factor (3.22) and introduce the symbol R for one of the factors.

$$\rho_{n+1} \leq C\left[\left(\prod_{k=r-m+1}^{r} e_n^k\right)\prod_{i=r-m+1}^{r}\cdot\left(C_{m+i}\prod_{l=r-m-i+2}^{r} e_{n-1}^l\right)R\right]^{\frac{1}{2}}. \tag{3.23}$$

We next absorb R and the C_{m+i} into C and break the middle product up to get

$$\rho_{n+1} \leq C\left[\left(\prod_{k=r-m+1}^{r} e_n^k\right)\left(\prod_{l=2-m}^{r} e_{n-1}^l\right)^m \times\prod_{i=n-m+1}^{r-1}\prod_{l=n-m-i+2}^{r} e_{n-1}^l\right]^{\frac{1}{2}}. \tag{3.24}$$

In the first product, we bound e_n^k by ρ_n. In the second product, we absorb the terms with nonpositive l into C and bound the remaining e_{n-1}^l by ρ_{n-1}. The last two products are absorbed into C. This gives

$$\rho_{n+1} \leq C\rho_n^{m/2}\rho_{n-1}^{rm/2}, \tag{3.25}$$

which is (3.19).

Derivation of (3.20) *from* (3.18). This derivation proceeds similarly to the previous one with factoring, a bootstrap step, absorption of terms into the leading constant, and estimating e_n^j by ρ_n. We will do the case $r \mid m$ and sketch the steps without comment. We write (3.18) as

$$\begin{aligned}
\rho_{n+1} &\leq C\left[\sum_{j=1}^{r}\left(C_{m+j}\prod_{i=1}^{n}\prod_{k=0}^{m/r-1} e_{n-k}^i\prod_{l=0}^{j} e_{n-m/r}^{r-l}\right)^2\right]^{\frac{1}{2}}\\
&= C\left[\left(\prod_{k=0}^{m/r-1}\prod_{i=0}^{r} e_{n-k}^i\right)\left(\prod_{k=0}^{m/r-1}\prod_{i=1}^{r}\left\{\prod_{\alpha=1}^{r}\prod_{\beta=1}^{m/r} e_{n-k-\beta}^{\alpha}\right.\right.\right.\\
&\quad\left.\left.\left.\times\prod_{\gamma=0}^{i} e_{n-k-m/r-1}^{r-\gamma}\right\}\right)\sum_{j=1}^{r}\left(C_{m+j}\prod_{l=0}^{j} e_{n-m/r}^{r-l}\right)^2\right]^{\frac{1}{2}}\\
&\leq C\left[\prod_{k=0}^{m/r-1}\rho_{n-k}^{r}\left(\prod_{k=0}^{m/r-1}\prod_{i=1}^{r}\left\{\prod_{\beta=1}^{m/r}\rho_{n-k-\beta}^{r}\rho_{n-k-m/r-1}^{i}\right\}\right)\right]^{\frac{1}{2}}\\
&\leq C\left(\prod_{k=0}^{m/r-1}\rho_{n-k}^{r/2}\right)\left(\prod_{k=0}^{m/r-1}\prod_{\beta=1}^{m/r}\rho^{n^2/2}\right)
\end{aligned} \tag{3.26}$$

which is (3.20). Inspection of this derivation shows that if r does not divide m, the steps are correct if m/r is replaced by $[m/r]$. The extra multiplicative terms, which will appear and which are omitted, may (and legitimately so) be considered to be absorbed into C.

4. Discussion and numerical results

The order of convergence enables us to make asymptotic comparisons (large n and large r) between the methods. Let two methods be used to approximate a root. Let these methods be characterized by m_i, r_i, λ_i, $i = 1, 2$. Suppose the methods take n_1 and n_2 steps respectively to achieve an approximation of a certain accuracy. Let x_{n_1} be the vector iterate of the first method after n_1 iterations, and let x_{n_2} be the vector iterate of the second method after n_2 iterations. Since x_{n_1} and x_{n_2} are approximations to x of comparable accuracy, then $\rho(x_{n_1}, x)$ and $\rho(x_{n_2}, x)$ are approximately equal. Then from (3.2), we see that the following equality is approximately true:

$$C_1^{\lambda_1^{n_1}} = C_2^{\lambda_2^{n_2}}. \tag{4.1}$$

Then

$$n_1/n_2 = \log_{\lambda_1}\lambda_2 + 1/n_2\log_{\lambda_1}\left(\frac{\log C_2}{\log C_1}\right). \tag{4.2}$$

For example, if $m = 2$ for both methods and method 2 uses r processors while method 1 uses one processor, (4.2) becomes

$$n_1/n_2 \geq \log_{(1+\sqrt{5})/2}\left(\frac{1+\sqrt{1+4r}}{2}\right) + \frac{\text{const}}{n_2}. \tag{4.3}$$

Here we have used the lower estimate for λ_2 derivable from (3.11).

Table 1 A comparison of three methods for the function $\sum_1^5 x^n$ (see text).

n	$r = 1$ $m = 2$	$r = 2$ $m = 2$		$r = 3$ $m = 3$		
0	-1.0		-1.0			-1.0
1	$.9$	0.9	$.1$	0.9	-0.8	$.7$
2	$-.6$	$.8 \times 10^{-1}$	$.8 \times 10^{-2}$	$-.8$	-1.5	$-.8$
3	$-.5$	$.6 \times 10^{-3}$	$.6 \times 10^{-4}$	$.2$	$-.2 \times 10^{-1}$	$-.2 \times 10^{-1}$
4	$.9 \times 10^{-1}$	$.4 \times 10^{-7}$	$.3 \times 10^{-9}$	$-.1 \times 10^{-1}$	$-.5 \times 10^{-1}$	$-.1 \times 10^{-3}$
5	$-.4 \times 10^{-1}$	$.8 \times 10^{-17}$	$.3 \times 10^{-17}$	$-.1 \times 10^{-3}$	$.1 \times 10^{-4}$	$-.1 \times 10^{-8}$
6	$-.6 \times 10^{-2}$			$.1 \times 10^{-8}$	$.2 \times 10^{-7}$	$.2 \times 10^{-16}$
7	$.1 \times 10^{-3}$					
8	$-.5 \times 10^{-6}$					
9	$-.7 \times 10^{-10}$					
10	$.3 \times 10^{-16}$					

For r and n_2 large, we have the asymptotic result

$$n_1/n_2 \geq \tfrac{1}{2} \log_{(1+\sqrt{5})/2} r + o(1). \tag{4.4}$$

We see then that while there is a gain in speed for parallel methods, it may be only logarithmic in the number of processors. A numerical experiment using three methods: $m = 2$ and $r = 1, 2, 3$ respectively, was tried out on the function $\sum_1^5 x^n$. The results are given in Table 1, where the entries are the vectors $\mathbf{x}_n$ with components displayed horizontally.

5. Remarks

Regula Falsi and the Lagrange interpolation polynomials were the basis of the algorithms in this paper. Any other iterative procedure for finding roots, such as Newton's method, could be treated and made into a parallel method along the lines presented here. For Newton's method the Hermite interpolation polynomials play the role of Lagrange's interpolation polynomials. Indeed, hybrid versions of a parallel method combining Lagrange and Hermite polynomials are possible.

At first sight, the orders of the methods presented here seem to contradict the known result that the order of convergence of a method using Lagrange polynomials varies between $(1 + \sqrt{5})/2$ (for Regula Falsi) and 2. For example, $\lambda = 3.59$ for $r = m = 3$ (see (3.12)). The explanation is that the order for the standard method assumes that one step in the algorithm corresponds to one evaluation of $f(x)$. One step in a parallel r-processor algorithm, however, corresponds to r evaluations of $f(x)$. Thus, the order of convergence of a parallel method should be discounted to $\lambda^{1/r}$. For the example cited $3.59^{1/3} = 1.53$. This number is less than the order of convergence of the usual sequential algorithm corresponding to $m = 3$ and $r = 1$. However, the sequential algorithm cannot even at the expense of computing power (i.e., more processors) be speeded up, whereas the parallel method can be speeded up.

Finally we point out that the higher order methods, which make up the components of the vector iterates, require more computations then lower order methods, even though all require one functional evaluation. We have not analyzed these costs.

References

1. L. Collatz, *Functional Analysis and Numerical Mathematics,* Academic Press, New York, 1966, (see especially Section 18).
2. J. Traub, *Iterative Methods for the Solution of Equations,* Prentice-Hall, Englewood Cliffs, New Jersey, 1964.

Received August 28, 1969