LMS J. Comput. Math. 18 (1) (2015) 713–729
doi:10.1112/S1461157015000236

A parallel root-finding algorithm

M. J. P. Nijmeijer

Abstract

We present a parallel algorithm to calculate a numerical approximation of a single, isolated root α of a function $f : \mathbb{R} \to \mathbb{R}$ which is sufficiently regular at and around α. The algorithm is derivative free and performs one function evaluation on each processor per iteration. It requires at least three processors and can be scaled up to any number of these. The order with which the generated sequence of approximants converges to α is equal to $(n + \sqrt{n^2 + 4})/2$ for $n + 1$ processors with $n \geqslant 2$. This assumes that particular combinations of the derivatives of f do not vanish at α.

1. *Introduction*

A root-finding algorithm is an iterative calculation scheme to approximate a single, isolated root of a function f [27]. The root α is a solution of the equation $f(\alpha) = 0$. We restrict ourselves to scalar, real-valued functions $f : \mathbb{R} \to \mathbb{R}$. Amongst the many different root-finding algorithms [1, 4, 6, 14, 20, 23–25], derivative-free algorithms do not require the calculation of a derivative of f. This is a desirable property in those cases where a derivative cannot be evaluated easily. Two classes of root-finding methods are interval methods and locally convergent methods. Interval methods [1, 4, 15] such as the bisection method [12] establish, at each iteration, an interval which must contain α. The length of the interval diminishes with each iteration and converges to zero. Locally convergent methods such as the secant method [9, 12, 21, 25] calculate an approximant of α at each iteration. These approximants converge to α if the starting approximant(s) is (are) close enough to α.

An efficient root-finding algorithm typically is required when the evaluation of $f(x)$ for a value x is relatively slow. In these cases we want to evaluate f as few times as possible, while still approximating α with a prescribed accuracy. The primary gauge for an algorithm's efficiency is its order of convergence [12, 25] (this assumes that the algorithm has a well-defined order of convergence, which is usually the case). An algorithm with a higher order of convergence requires fewer iterations if the required accuracy of the approximation is high enough. The fastest derivative-free methods with one function evaluation per iteration achieve orders of convergence asymptotically close to two [5, 17, 24, 27].

Algorithms with multiple function evaluations (so-called 'multi-point' algorithms [3, 7, 13, 22, 26, 27]) such as Steffensen's method [23] achieve orders of convergence two and higher. However, it takes more time to execute an iteration. To compare algorithms with different numbers of function evaluations per iteration, one often uses the 'effective order of convergence' [27]. If a method uses m function evaluations per iteration and achieves an order of convergence ϕ, the effective order is defined as $\phi^{1/m}$. We are not aware of any derivative-free algorithm with an effective order of convergence equal to or larger than two [28, 29].

So far we have discussed sequential algorithms, executing on a single processor core. The availability of multiple cores allows us to evaluate f for different values of x in parallel without paying a heavy time penalty. If we start with n different approximants x_i of α and we can

Received 8 February 2015; revised 15 June 2015.

2010 Mathematics Subject Classification 26A06, 39B22, 65Y05 (primary).

construct n new and better approximants x_i' from the calculated $f(x_i)$, we have, in principle, a parallel root-finding algorithm.

Most of the work on parallel root-finding algorithms for a single, scalar root was done in the seventies. One can easily write down a parallelization scheme for the bisection method. This was taken further by Gal and Miranker [**10**] who developed parallel schemes of the interval type for functions with known bounds on the slope. Work on a locally convergent algorithm was carried out by Corliss [**8**]. He developed a three-core algorithm which seeded each core i with an initial approximant x_i, calculated f in parallel for the three values x_i and then carried out a secant step on the combinations (x_1, x_2), (x_1, x_3) and (x_2, x_3) to obtain three new approximants. He showed that the algorithm converges with order two. He investigated what happens when this 'three-core secant method' is scaled up to an 'n-core secant method' but found that the order does not increase further (G. F. Corliss, Personal communication, 2015).

Contrary to the case of a single scalar root, there is an extensive literature on parallel algorithms to find all the roots of a polynomial [**11**, **19**]. Parallel algorithms for the solution of systems of non-linear equations have also gained much attention [**16**].

We present a parallelization scheme which is derivative free and locally convergent. It is a refinement of the algorithm of Corliss for the case of three processor cores. Its order of convergence is $1 + \sqrt{2} \approx 2.41$, in this case. In theory, our scheme can be scaled up to any number of cores with ever-growing order of convergence. In practice, the order of convergence quickly becomes so large that the root can be calculated to machine precision in a few iterations.

We explain the sequential algorithms that form the basis of our parallelization scheme in the next section. Section 3 discusses how we obtain n improved approximants from n starting approximants. This defines our parallel algorithm. Section 4 is of a more mathematical nature and derives the order of convergence and the asymptotic error term. Some alternatives to our algorithm are briefly discussed in § 5. Two types of numerical results are shown in this paper: results from a sequential algorithm which calculates the results of an n-core algorithm with high-precision arithmetic, and results from a truly parallel Java implementation of the three core algorithm. Numerical results for the Java implementation are presented in § 6. Concluding remarks are made in the last section.

2. *Sequential algorithms*

Suppose we have an open interval $I \subset \mathbb{R}$ and a function $f : I \to \mathbb{R}$. Suppose $\alpha \in I$ and $f(\alpha) = 0$. Our parallelization scheme is based on any sequential root-finding algorithm which calculates an approximant $a_m(x_1, \ldots, x_{m+2})$ of the root α from $m+2$ initial approximants $x_1, \ldots, x_{m+2}$. To calculate a_m, the function f has to be evaluated at all of the initial approximants x_i but not at any other points. All of the initial approximants are in I and it is assumed that a_m is in I as well. The approximant a_m must have the property

$$a_m(x_1, \ldots, x_{m+2}) = \alpha + \left\{ \prod_{i=1}^{m+2} (x_i - \alpha) \right\} A_m(x_1, \ldots, x_{m+2}), \tag{1}$$

where A_m is continuous in the point $(x_1, \ldots, x_{m+2}) = (\alpha, \ldots, \alpha) \equiv \boldsymbol{\alpha}_{m+2}$ if $f \in C^{m+2}(I)$ and $f^{(1)}(\alpha) \neq 0$. The continuity implies

$$\lim_{(x_1,\ldots,x_{m+2}) \to \boldsymbol{\alpha}_{m+2}} A_m(x_1, \ldots, x_{m+2}) = A_{\infty m}. \tag{2}$$

The following are sequential algorithms with these characteristics.

(i) *Direct polynomial interpolation*. With this method, we fit a polynomial P of degree $m+1$ to the points $\{(x_i, f(x_i))\}_{i=1}^{m+2}$ and calculate a_m as an appropriately selected root of P.

https://doi.org/10.1112/S1461157015000236 Published online by Cambridge University Press

If we denote $A_{\infty m}$ for this method as $A_{\infty m}^{\mathrm{DP}}$, then (see [27])

$$A_{\infty m}^{\mathrm{DP}} = \frac{(-1)^{m+2}}{(m+2)!}\frac{f^{(m+2)}(0)}{f^{(1)}(0)}. \tag{3}$$

We denote the ith derivative of the function f at the point x by $f^{(i)}(x)$. See § 4.2 of Traub [27] for a detailed treatment of both direct and inverse polynomial approximation.

(ii) *Inverse polynomial interpolation.* With this method, we fit a polynomial $\mathcal{P}$ of degree $m+1$ to the points $\{(f(x_i), x_i)\}_{i=1}^{m+2}$. That is, we fit a polynomial $\mathcal{P}$ to $\mathcal{F}$, the inverse function of f on I. We calculate a_m as $\mathcal{P}(0)$. If we denote $A_{\infty m}$ for this method as $A_{\infty m}^{\mathrm{IP}}$, then (see [27])

$$A_{\infty m}^{\mathrm{IP}} = -\frac{(-1)^{m+2}}{(m+2)!}\frac{\mathcal{F}^{(m+2)}(0)}{\{\mathcal{F}^{(1)}(0)\}^{m+2}}. \tag{4}$$

Examples for $A_{\infty m}^{\mathrm{IP}}$ are

$$\begin{gathered} A_{\infty 0}^{\mathrm{IP}} = C_2, \quad A_{\infty 1}^{\mathrm{IP}} = -C_3 + 2C_2^2, \quad A_{\infty 2}^{\mathrm{IP}} = C_4 - 5C_2C_3 + 5C_2^3, \\ A_{\infty 3}^{\mathrm{IP}} = -C_5 + 6C_2C_4 + 3C_3^2 - 21C_2^2C_3 + 14C_2^4, \end{gathered} \tag{5}$$

where $C_n = f^{(n)}(\alpha)/(n!f^{(1)}(\alpha))$.

(iii) *The method of improved approximants.* With this method, a_m can be calculated recursively as in [17]

$$a_m(x_1, \ldots, x_{m+2}) = \frac{a_{m-1}(x_1, \ldots, x_{m+1})x_{m+2} - x_1 a_{m-1}(x_2, \ldots, x_{m+2})}{a_{m-1}(x_1, \ldots, x_{m+1}) + x_{m+2} - x_1 - a_{m-1}(x_2, \ldots, x_{m+2})}, \tag{6}$$

where the recursion terminates at the secant step a_0: that is

$$a_0(x_1, x_2) = x_1 - f(x_1)\frac{x_1 - x_2}{f(x_1) - f(x_2)}. \tag{7}$$

If we denote $A_{\infty m}$, in this case, as $A_{\infty m}^{\mathrm{IA}}$, then

$$A_{\infty m}^{\mathrm{IA}} = \frac{(-1)^{m+1}}{(m+1)!}f^{(1)}(\alpha)\left(\frac{\partial^{m+1}}{\partial x^{m+1}}\frac{1}{f[\alpha, x]}\right)_{x=\alpha}, \tag{8}$$

where $f[\alpha, x] = (f(\alpha) - f(x))/(\alpha - x)$ is the divided difference of f. Examples for $A_{\infty m}^{\mathrm{IA}}$ are

$$\begin{gathered} A_{\infty 0}^{\mathrm{IA}} = C_2, \quad A_{\infty 1}^{\mathrm{IA}} = -C_3 + C_2^2, \quad A_{\infty 2}^{\mathrm{IA}} = C_4 - 2C_2C_3 + C_2^3, \\ A_{\infty 3}^{\mathrm{IA}} = -C_5 + 2C_2C_4 + C_3^2 - 3C_2^2C_3 + C_2^4. \end{gathered} \tag{9}$$

In choosing between the method of direct polynomial interpolation and inverse polynomial interpolation, we prefer the latter. Inverse polynomial interpolation avoids the need to calculate the root(s) of a polynomial of a possibly high degree. Our numerical examples, therefore, are based on inverse polynomial interpolation and the method of improved approximants.

All three methods reduce to the secant method for $m = 0$.

Because of (1), we expect that higher values of m, in general, yield better approximations of α. The ratio

$$\frac{a_m(x_1, \ldots, x_{m+2}) - \alpha}{a_{m-1}(x_1, \ldots, x_{m+1}) - \alpha} = (x_{m+2} - \alpha)\frac{A_m(x_1, \ldots, x_{m+2})}{A_{m-1}(x_1, \ldots, x_{m+1})} \tag{10}$$

can be made arbitrarily small by taking x_{m+2} close enough to α provided that A_m is bounded around $\boldsymbol{\alpha}_{m+2}$ and A_{m-1} is bounded from below around $\boldsymbol{\alpha}_{m+1}$. The first condition is guaranteed if $f \in C^{m+2}(I)$ and $f^{(1)}(\alpha) \neq 0$. However, the second condition is not guaranteed.

https://doi.org/10.1112/S1461157015000236 Published online by Cambridge University Press

3. *The parallel algorithm*

Suppose we have $n+2$ processor cores at our disposal with $n \geqslant 1$, that is, we have at least three cores. The parallel algorithm can be described as follows.

(1) Seed each processor i with an initial estimate x_i.
(2) Do the following in parallel for all processors i.
  (a) Calculate $f(x_i)$.
  (b) Broadcast $f(x_i)$ to all other processors.
  (c) Calculate $a_n(x_1, \ldots, x_{n+2})$ on processor 1 as the new x_1. Calculate $a_{n-1}(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_{n+2})$ on processor i as the new x_i for $i = 2, \ldots, n+2$.
  (d) Broadcast x_i to all other processors.
(3) Repeat step 2 until x_1 is a sufficiently close approximation of α.

This describes the entire algorithm. We make the following remarks.

(i) The a_n and a_{n-1} are expected to be the most accurate new approximants we can construct from the starting values $x_1, \ldots, x_{n+2}$ (compared with a_m for $m < n-1$).
(ii) We expect a_n to be a better approximation to α than any of the a_{n-1}. Therefore we expect x_1 to be a better approximation than $x_2, \ldots, x_{n+1}$ in all iterations. Hence, of the $n+2$ possible approximants a_{n-1} that we can calculate from the set $\{x_1, \ldots, x_{n+2}\}$, we pick those $n+1$ that have x_1 as one of their arguments.
(iii) If we have only two calculation cores available, the only improved approximant we can calculate is $a_0(x_1, x_2)$. Since this does not allow us to continue with the next iteration, we need at least three cores. In the case of three cores, we continue with $a_1(x_1, x_2, x_3)$, $a_0(x_1, x_2)$ and $a_0(x_1, x_3)$.

We carried out numerical computations with a sequential code which simulated the multi-core case (Tables 1, B1 and B2) and with a truly parallel code. The simulations of the multi-core case are to obtain high-precision results with which we verify the mathematical predictions of § 4. This code is written in the open-source program Sage (www.sagemath.org) which allows for computations with an arbitrarily high precision.

We have also written a truly parallel program in Java (Tables 2–4). It demonstrates the feasibility of turning the algorithm into a working parallel program. It also serves to obtain some idea of its performance, although a true study of the performance characteristics is not what we are aiming at. The parallel implementation is discussed in § 6.

TABLE 1. *Comparison of the convergence of the secant algorithm with the three-core parallel algorithm for the function f. The parallel algorithm uses the method of improved approximants. The secant sequence develops as $x_p = a_0(x_{p-1}, x_{p-2})$ with starting values $x_{-1} = -0.1$ and $x_0 = 0.1$. The development of the three sequences generated by the parallel algorithm is described in § 3. Starting values for the parallel algorithm are $x_{0,1} = -0.1$, $x_{0,2} = 0.1$ and $x_{0,3} = 0.2$.*

$$f(x) = \frac{x(x^2 + x - 1)}{x + 1}$$

	Secant algorithm	Parallel algorithm		
p	x_p	$x_{p,1}$	$x_{p,2}$	$x_{p,3}$
-1	-0.1			
0	0.1	-0.1	0.1	0.2
1	1.99×10^{-2}	-9.33×10^{-3}	4.66×10^{-2}	1.99×10^{-2}
2	-4.88×10^{-3}	-2.87×10^{-5}	3.77×10^{-4}	9.22×10^{-4}
3	1.99×10^{-4}	-3.00×10^{-11}	5.30×10^{-8}	2.17×10^{-8}
4	1.92×10^{-6}	-1.03×10^{-25}	1.30×10^{-18}	3.18×10^{-18}
5	-7.65×10^{-10}	-1.28×10^{-60}	6.57×10^{-43}	2.68×10^{-43}

https://doi.org/10.1112/S1461157015000236 Published online by Cambridge University Press

Table 1 compares the secant algorithm with the three-core parallel algorithm. The function f used in this example is shown in the table header. It has the root $\alpha = 0$. The table shows the sequence of approximants generated by the secant algorithm, together with the three sequences $\{x_{p,i}\}$ generated by core i, for the first five iterations. The method of improved approximants was used for the parallel algorithm. The sequences of the parallel algorithm develop as $x_{p,1} = a_1(x_{p-1,1}, x_{p-1,2}, x_{p-1,3})$, $x_{p,2} = a_0(x_{p-1,1}, x_{p-1,3})$ and $x_{p,3} = a_0(x_{p-1,1}, x_{p-1,2})$. The table shows that $x_{p,1}$ is a much better approximation of the root than $x_{p,2}$ or $x_{p,3}$. After five iterations, the parallel algorithm approximates the root with an error of $\sim 10^{-60}$ whereas the secant algorithm 'only' comes to within $\sim 10^{-10}$.

The difference in convergence speed is a manifestation of the larger order of convergence of the parallel algorithm. The secant algorithm has an order of convergence $(1 + \sqrt{5})/2 \approx 1.62$ (assuming that particular conditions on f are met, which is the case for the function in Table 1). The order of convergence of the parallel algorithm will be calculated in the next section. We will see that the sequence $\{x_{p,1}\}$ of the three-core parallel algorithm converges with order $1 + \sqrt{2} \approx 2.41$.

4. *Convergence properties*

4.1. *Basic convergence*

Call $x_{p,i}$ the approximant calculated on core i in iteration p. We have $n + 2$ cores. The set $\{x_{p,1}, \ldots, x_{p,n+2}\}$ develops as

$$\begin{aligned} x_{p,1} &= a_n(x_{p-1,1}, \ldots, x_{p-1,n+2}) \\ x_{p,i} &= a_{n-1}(x_{p-1,1}, \ldots, x_{p-1,i-1}, x_{p-1,i+1}, \ldots, x_{p-1,n+2}) \quad i = 2, \ldots, n+2 \end{aligned} \tag{11}$$

for $p = 1, 2, 3, \ldots$ with starting values $\{x_{0,1}, \ldots, x_{0,n+2}\}$.

One conveniently describes the development in the coordinate frame Y in which all distances are measured with respect to α: $y = x - \alpha$, $f_Y(y) = f(x)$, and $a_{Yn}(y_1, \ldots, y_{n+2}) = a_n(x_1, \ldots, x_{n+2}) - \alpha$. The root is at $y = 0$ in this coordinate frame: $f_Y(0) = 0$.

From (1)

$$\begin{aligned} y_{p,1} &= \left\{ \prod_{j=1}^{n+2} y_{p-1,j} \right\} A_{Yp,1}, \\ y_{p,i} &= \left\{ \prod_{\substack{j=1 \\ j \neq i}}^{n+2} y_{p-1,j} \right\} A_{Yp,i}, \quad i = 2, \ldots, n+2 \end{aligned} \tag{12}$$

with

$$\begin{aligned} A_{Yp,1} &= A_{Yn}(y_{p-1,1}, \ldots, y_{p-1,n+2}) \\ A_{Yp,i} &= A_{Yn-1}(y_{p-1,1}, \ldots, y_{p-1,i-1}, y_{p-1,i+1}, \ldots, y_{p-1,n+2}). \end{aligned} \tag{13}$$

The starting values are $\{y_{0,1}, \ldots, y_{0,n+2}\}$. This allows us to establish the basic convergence properties in the following lemma.

LEMMA 1. *Let I be an open interval of real values and f a function $f : I \to \mathbb{R}$ with $f \in C^{n+2}(I)$. Let $\alpha \in I$, $f(\alpha) = 0$ and $f^{(1)}(\alpha) \neq 0$. Then there exists an $\epsilon > 0$ such that the sequences $\{x_{p,1}\}_{p=0}^{\infty}, \ldots, \{x_{p,n+2}\}_{p=0}^{\infty}$ generated by the parallel algorithm on $n + 2$ processor cores all converge to α if the starting values $x_{0,1}, \ldots, x_{0,n+2}$ are all within a distance ϵ of α.*

https://doi.org/10.1112/S1461157015000236 Published online by Cambridge University Press

Proof. Under the conditions of the lemma, there is an $\epsilon' > 0$ such that the $A_{Y\,p,i}$ are bounded by $\overline{A}$ for all $i = 1, \ldots, n+2$ if $|y_{p-1,i}| < \epsilon'$ for all i. This follows from the properties of A_m mentioned in § 2.

Take $\epsilon < \min(\epsilon', \overline{A}^{-1/n}, \overline{A}^{-1/(n+1)})$ and $|y_{p-1,i}| < \epsilon$ for all $i = 1, \ldots, n+2$. Then

$$|y_{p,1}| = |y_{p-1,1}| \left\{ \prod_{j=2}^{n+2} |y_{p-1,j}| \right\} |A_{p,1}| < |y_{p-1,1}| \epsilon^{n+1} \overline{A} < |y_{p-1,1}| \tag{14}$$

and

$$|y_{p,i}| = \left\{ \prod_{\substack{j=1 \\ j \neq i}}^{n+2} |y_{p-1,j}| \right\} |A_{p,i}| < \epsilon^{n+1} \overline{A} = \epsilon \epsilon^{n} \overline{A} < \epsilon \tag{15}$$

for $i = 2, \ldots, n+2$. Hence $|y_{p,i}| < \epsilon$ for all $i = 1, \ldots, n+2$ and $|y_{p,1}| < |y_{p-1,1}|$. We can continue the argument recursively and see that the sequence $\{|y_{p,1}|\}$ is decreasing and therefore converges.

To show that the sequence converges to zero, reason as follows. Suppose $\lim_{p\to\infty} |y_{p,1}| = a > 0$. Then from (14),

$$\lim_{p\to\infty} \left\{ \prod_{j=2}^{n+2} |y_{p-1,j}| \right\} |A_{p,1}| = 1.$$

Since $\prod_{j=2}^{n+2} |y_{p-1,j}| < \epsilon^{n+1}$, there must be a $p_{\min}$ such that $|A_{p,1}| > \epsilon^{-(n+1)}$ for $p > p_{\min}$. This implies that $\epsilon^{-(n+1)} < \overline{A}$, from which follows $\epsilon > \overline{A}^{-1/(n+1)}$. This contradicts our choice of ϵ, and therefore we conclude that $\lim_{p\to\infty} |y_{p,1}| = 0$.

Having shown that $|y_{p,1}|$ converges to zero, we easily see that the other sequences also converge to zero from

$$|y_{p,i}| = |y_{p,1}| \left\{ \prod_{\substack{j=2 \\ j \neq i}}^{n+2} |y_{p-1,j}| \right\} |A_{p,i}| < |y_{p,1}| \epsilon^{n} \overline{A}.$$

This concludes our proof. □

Lemma 3 in [17] states that instead of the condition $f \in C^{n+2}(I)$, the two conditions $f \in C^{n+1}(I)$ and $f^{(n+1)}$ being Lipschitz continuous on I will already guarantee the boundedness of A_n in the case of the method of improved approximants. Hence we can weaken the condition $f \in C^{n+2}(I)$ in Lemma 1 to these two conditions in the case where the parallel algorithm is implemented with improved approximants. The same is possibly true for an implementation with direct or inverse polynomial interpolation, but this has not been investigated.

4.2. *Order of convergence*

Having established the basic convergence of the algorithm, the theorem below states our most detailed result on the convergence properties.

THEOREM 1. *Let I be an open interval of real values and f a function $f : I \to \mathbb{R}$ with $f \in C^{n+2}(I)$. Let $\alpha \in I$, $f(\alpha) = 0$, and $f^{(1)}(\alpha) \neq 0$. Let $A_{\infty n} \neq 0$ and $A_{\infty n-1} \neq 0$. Then there exists an $\epsilon > 0$ such that the sequence $\{x_{p,1}\}_{p=0}^{\infty}$ generated by the parallel algorithm on $n+2$ processor cores converges to α with order $\phi = (n+1+\sqrt{(n+1)^2+4})/2$: that is*

$$\lim_{p\to\infty} \frac{|x_{p,1} - \alpha|}{|x_{p-1,1} - \alpha|^{\phi}} = |A_{\infty n}|^{-(\phi^2 - 2\phi - 1)/(\phi+1)} |A_{\infty n-1}|^{\phi - 1}$$

if the starting values $x_{0,1}, \ldots, x_{0,n+2}$ are all within a distance ϵ of α.

https://doi.org/10.1112/S1461157015000236 Published online by Cambridge University Press

The combination $\phi^2 - 2\phi - 1$ is equal to $(n-1)\phi$. The limit

$$|A_{\infty n}|^{-(\phi^2-2\phi-1)/(\phi+1)}|A_{\infty n-1}|^{\phi-1}$$

is called the asymptotic error [**12**].

Proof. The recursive development of the approximants as prescribed by (12) implies that the approximants take the form

$$y_{p,i} = \left\{\prod_{j=1}^{n+2} y_{0,j}^{k_{p,i,j}}\right\}\mathcal{A}_{p,i}, \quad i = 1, \ldots, n+2. \tag{16}$$

Substituting these forms into (12) gives recursion relations for the powers $k_{p,i,j}$ and the coefficients $\mathcal{A}_{p,i}$. The recursions for the powers can be written in matrix form as

$$\begin{pmatrix} k_{p,1,j} \\ k_{p,2,j} \\ \vdots \\ k_{p,n+2,j} \end{pmatrix} = \vec{\vec{M}} \begin{pmatrix} k_{p-1,1,j} \\ k_{p-1,2,j} \\ \vdots \\ k_{p-1,n+2,j} \end{pmatrix}, \quad \vec{\vec{M}} = \begin{pmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & 0 & 1 & \ldots & 1 \\ 1 & 1 & 0 & \ldots & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & 1 & \ldots & 0 \end{pmatrix} \tag{17}$$

with starting values $k_{0,i,j} = \delta_{i,j}$, where $\delta_{i,j}$ is the Kronecker delta. The matrix $\vec{\vec{M}}$ has $n+2$ rows and $n+2$ columns.

The coefficients $\mathcal{A}_{p,i}$ satisfy the recursions

$$\begin{aligned} \mathcal{A}_{p,1} &= \left\{\prod_{j=1}^{n+2} \mathcal{A}_{p-1,j}\right\} A_{Yp,1}, \\ \mathcal{A}_{p,i} &= \left\{\prod_{\substack{j=1 \\ j\neq i}}^{n+2} \mathcal{A}_{p-1,j}\right\} A_{Yp,i}, \quad i = 2, \ldots, n+2, \end{aligned} \tag{18}$$

for $p = 1, 2, 3, \ldots$ with start values $\mathcal{A}_{0,i} = 1$ for all $i = 1, \ldots, n+2$. The coefficients take the form

$$\begin{aligned} \mathcal{A}_{p,1} &= \prod_{j=1}^{p} A_{Yj,1}^{\beta_{p-j}} \prod_{k=2}^{n+2} A_{Yj,k}^{\theta_{p-j}}, \\ \mathcal{A}_{p,i} &= \prod_{j=1}^{p} A_{Yj,1}^{\overline{\beta}_{p-j}} A_{Yj,i}^{\vartheta_{p-j}} \prod_{\substack{k=2 \\ k\neq i}}^{n+2} A_{Yj,k}^{\overline{\theta}_{p-j}}. \end{aligned} \tag{19}$$

Substitution of this form into (18) results in the recursion relations and start values for the powers: that is

$$\begin{aligned} \begin{pmatrix} \beta_p \\ \overline{\beta}_p \end{pmatrix} &= \begin{pmatrix} 1 & n+1 \\ 1 & n \end{pmatrix} \begin{pmatrix} \beta_{p-1} \\ \overline{\beta}_{p-1} \end{pmatrix}, \qquad \begin{pmatrix} \beta_0 \\ \overline{\beta}_0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \\ \begin{pmatrix} \vartheta_p \\ \theta_p \\ \overline{\theta}_p \end{pmatrix} &= \begin{pmatrix} 0 & 1 & n \\ 1 & 1 & n \\ 1 & 1 & n-1 \end{pmatrix} \begin{pmatrix} \vartheta_{p-1} \\ \theta_{p-1} \\ \overline{\theta}_{p-1} \end{pmatrix}, \qquad \begin{pmatrix} \vartheta_0 \\ \theta_0 \\ \overline{\theta}_0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}. \end{aligned} \tag{20}$$

https://doi.org/10.1112/S1461157015000236 Published online by Cambridge University Press

Solution of the recursion relations. We study the recursion relations (17) and (20). The solution of (17) is

$$\begin{pmatrix} k_{p,1,j} \\ k_{p,2,j} \\ \vdots \\ k_{p,n+2,j} \end{pmatrix} = (\vec{\vec{M}})^p \begin{pmatrix} \delta_{1,j} \\ \delta_{2,j} \\ \vdots \\ \delta_{n+2,j} \end{pmatrix}. \tag{21}$$

To take the matrix to the power p, we have to diagonalize it. Its characteristic equation is

$$(\lambda+1)^n(\lambda^2-(n+1)\lambda-1)=0.$$

Its eigenvalues are therefore

$$\begin{aligned} \lambda_1 &= \tfrac{1}{2}\{n+1+\sqrt{(n+1)^2+4}\}, \\ \lambda_2 = \lambda_3 &= \ldots = \lambda_{n+1} = -1, \\ \lambda_{n+2} &= \tfrac{1}{2}\{n+1-\sqrt{(n+1)^2+4}\}. \end{aligned} \tag{22}$$

It is easy to see that $\lambda_1 > 1$ for $n \geqslant 1$. With $\lambda_1\lambda_{n+2} = -1$, this gives $-1 < \lambda_{n+2} < 0$. An orthogonal set of eigenvectors is

$$\begin{aligned} \vec{v}_i &= \left(\frac{\lambda_i+1}{\lambda_i}, \underbrace{1,\ldots,1}_{n+1 \text{ times}}\right), \quad i = 1, n+2, \\ \vec{v}_i &= \left(0, \underbrace{\frac{1}{i-1},\ldots,\frac{1}{i-1}}_{i-1 \text{ times}}, -1, \underbrace{0,\ldots,0}_{n-i+1 \text{ times}}\right), \quad i = 2,\ldots,n+1, \end{aligned} \tag{23}$$

where $\vec{v}_i$ is the eigenvector corresponding to eigenvalue λ_i. With the eigenvalues and eigenvectors, it is possible to diagonalize the matrix $\vec{\vec{M}}$ and calculate the $k_{p,i,j}$: that is

$$\begin{aligned} k_{p,i,j} = (-1)^p\delta_{i,j} &+ \frac{1}{\vec{v}_1\cdot\vec{v}_1}\{\lambda_1^p + (-1)^{p+1}\}(\vec{v}_1)_i(\vec{v}_1)_j \\ &+ \frac{1}{\vec{v}_{n+2}\cdot\vec{v}_{n+2}}\{\lambda_{n+2}^p + (-1)^{p+1}\}(\vec{v}_{n+2})_i(\vec{v}_{n+2})_j, \end{aligned} \tag{24}$$

where $(\vec{v})_i$ is the ith element of vector $\vec{v}$, and

$$\vec{v}_i\cdot\vec{v}_i = \frac{(\lambda_i+1)^2}{\lambda_i^2} + n + 1 = \frac{(\lambda_i+1)(\lambda_i^2+1)}{\lambda_i^2}, \quad i = 1, n+2. \tag{25}$$

Diagonalization of the matrices in (20) is straightforward. We obtain

$$\begin{aligned} \beta_p &= \frac{1}{\lambda_1-\lambda_{n+2}}\{(\lambda_1+1)\lambda_1^{p-1} - (\lambda_{n+2}+1)\lambda_{n+2}^{p-1}\}, \\ \overline{\beta}_p &= \frac{1}{\lambda_1-\lambda_{n+2}}\{\lambda_1^p - \lambda_{n+2}^p\}, \\ \vartheta_p &= (-1)^p + \frac{1}{\lambda_1+\lambda_{n+2}}\left\{(-1)^{p+1} + \frac{1}{\lambda_1-\lambda_{n+2}}((\lambda_1-1)\lambda_1^p - (\lambda_{n+2}-1)\lambda_{n+2}^p)\right\}, \\ \theta_p &= \frac{1}{\lambda_1-\lambda_{n+2}}\{\lambda_1^p - \lambda_{n+2}^p\}, \\ \overline{\theta}_p &= \frac{1}{\lambda_1+\lambda_{n+2}}\left\{(-1)^{p+1} + \frac{1}{\lambda_1-\lambda_{n+2}}((\lambda_1-1)\lambda_1^p - (\lambda_{n+2}-1)\lambda_{n+2}^p)\right\}. \end{aligned} \tag{26}$$

https://doi.org/10.1112/S1461157015000236 Published online by Cambridge University Press

Properties of the sequence $\{y_{p,1}\}$. We use the results for the various powers to study the convergence of the sequence $\{y_{p,1}\}_{p=0}^{\infty}$. We focus on the sequence for $y_{p,1}$ because it is a closer approximant of the root than $y_{p,i}$ with $i > 1$. Using (16) we write

$$|y_{p,1}| = F_p|\mathcal{A}_{p,1}|, \quad F_p = \prod_{j=1}^{n+2} |y_{0,j}|^{k_{p,1,j}}. \tag{27}$$

We first study F_p. To simplify the full form (24), we write

$$k_{p,1,j} = (-1)^p\delta_{1j} + c_j\{\lambda_1^p + (-1)^{p+1}\} + d_j\{\lambda_{n+2}^p + (-1)^{p+1}\}, \tag{28}$$

with

$$c_j = \frac{1}{\vec{v}_1 \cdot \vec{v}_1}\frac{\lambda_1 + 1}{\lambda_1}(\vec{v_1})_j = \begin{cases} \dfrac{\lambda_1 + 1}{\lambda_1^2 + 1} & \text{if } j = 1, \\ \dfrac{\lambda_1}{\lambda_1^2 + 1} & \text{if } j = 2, \ldots, n+2, \end{cases} \tag{29}$$

and the same expressions for d_j but with λ_1 replaced by λ_{n+2}. The simplified form (28) gives, for F_p,

$$F_p = |y_{0,1}|^{(-1)^p} \prod_{j=1}^{n+2} |y_{0,j}|^{c_j\{\lambda_1^p + (-1)^{p+1}\} + d_j\{\lambda_{n+2}^p + (-1)^{p+1}\}}, \tag{30}$$

and

$$\begin{aligned} \frac{F_p}{F_{p-1}^{\lambda_1}} &= |y_{0,1}|^{(-1)^p - \lambda_1(-1)^{p-1}} \\ &\quad \times \prod_{j=1}^{n+2} |y_{0,j}|^{c_j\{(-1)^{p+1} - \lambda_1(-1)^p\} + d_j\{(\lambda_{n+2} - \lambda_1)\lambda_{n+2}^{p-1} + (-1)^{p+1} - \lambda_1(-1)^p\}}. \end{aligned} \tag{31}$$

With $c_1 + d_1 = 1$ and $c_j + d_j = 0$ for $j = 2, \ldots, n+2$, this reduces to

$$\frac{F_p}{F_{p-1}^{\lambda_1}} = \prod_{j=1}^{n+2} |y_{0,j}|^{d_j(\lambda_{n+2} - \lambda_1)\lambda_{n+2}^{p-1}}, \tag{32}$$

giving

$$\lim_{p\to\infty} \frac{F_p}{F_{p-1}^{\lambda_1}} = 1. \tag{33}$$

We now turn our attention to $\mathcal{A}_{p,1}$ in (27). From (19) in combination with the identities $\beta_p - \lambda_1\beta_{p-1} = (\lambda_{n+2} + 1)\lambda_{n+2}^{p-2}$, $\theta_p - \lambda_1\theta_{p-1} = \lambda_{n+2}^{p-1}$, $\beta_1 - \lambda_1\beta_0 = 1 - \lambda_1$, and $\theta_1 - \lambda_1\theta_0 = 1$,

$$\frac{|\mathcal{A}_{p,1}|}{|\mathcal{A}_{p-1,1}|^{\lambda_1}} = |A_{Yp,1}||A_{Yp-1,1}|^{1-\lambda_1}\left\{\prod_{k=2}^{n+2} |A_{Yp-1,k}|\right\}\mathcal{Q}_{p-2}, \tag{34}$$

with

$$\mathcal{Q}_p = \prod_{j=1}^{p} |A_{Yj,1}|^{(1+\lambda_{n+2})\lambda_{n+2}^{p-j}}\left\{\prod_{k=2}^{n+2} |A_{Yj,k}|\right\}^{\lambda_{n+2}^{p-j+1}}. \tag{35}$$

Under the conditions of the theorem we know that all sequences $\{y_{p,i}\}_{p=0}^{\infty}$ generated by the algorithm converge to zero (cf. Lemma 1). We also know that A_{Yn} is continuous in the point $\mathbf{0}_{n+2}$, while A_{Yn-1} is continuous in the point $\mathbf{0}_{n+1}$ (cf. (2)). Hence $\lim_{p\to\infty} A_{Yp,1} = A_{\infty n}$ and

https://doi.org/10.1112/S1461157015000236 Published online by Cambridge University Press

$\lim_{p\to\infty} A_{Yp,k} = A_{\infty n-1}$ for all $k = 2,\ldots,n+2$. Assuming that both limits are not equal to zero, we write

$$A_{Yj,1} = A_{\infty n}(1+\delta_{j,1}), \quad A_{Yj,k} = A_{\infty n-1}(1+\delta_{j,k}), \tag{36}$$

for $k = 2,\ldots,n+2$. We have $\lim_{p\to\infty}\delta_{p,i} = 0$ for all $i = 1,\ldots,n+2$. If we choose the starting values $y_{0,1},\ldots,y_{0,n+2}$ close enough to the root zero, we have $|\delta_{j,i}| < 1$ for all $j \geqslant 1$ and $i = 1,\ldots,n+2$. Substituting (36) into (35), $\mathcal{Q}_p$ takes the form

$$\mathcal{Q}_p = |A_{\infty n}|^{(1+\lambda_{n+2})(1-\lambda_{n+2}^p)/(1-\lambda_{n+2})}|A_{\infty n-1}|^{-(1+\lambda_{n+2})(1-\lambda_{n+2}^p)}\mathcal{R}_p, \tag{37}$$

with

$$\begin{aligned}\mathcal{R}_p &= \prod_{j=1}^{p}|1+\delta_{j,1}|^{(1+\lambda_{n+2})\lambda_{n+2}^{p-j}}\left\{\prod_{k=2}^{n+2}|1+\delta_{j,k}|\right\}^{\lambda_{n+2}^{p-j+1}} = \prod_{j=1}^{p}\tau_j^{\lambda_{n+2}^{p-j}},\\ \tau_j &= |1+\delta_{j,1}|^{(1+\lambda_{n+2})}\left\{\prod_{k=2}^{n+2}|1+\delta_{j,k}|\right\}^{\lambda_{n+2}}\end{aligned} \tag{38}$$

where $\tau_j > 0$ for all j, and $\lim_{j\to\infty}\tau_j = 1$. Note that $\mathcal{R}_p$ satisfies the recursion relation $\mathcal{R}_p = \tau_p\mathcal{R}_{p-1}^{\lambda_2}$ with $\mathcal{R}_0 = 1$. Based on this recursion, we prove in Appendix A that $\lim_{p\to\infty}\mathcal{R}_p = 1$, from which follows

$$\lim_{p\to\infty}\mathcal{Q}_p = |A_{\infty n}|^{(1+\lambda_{n+2})/(1-\lambda_{n+2})}|A_{\infty n-1}|^{-(1+\lambda_{n+2})}, \tag{39}$$

and

$$\lim_{p\to\infty}\frac{|\mathcal{A}_{p,1}|}{|\mathcal{A}_{p-1,1}|^{\lambda_1}} = |A_{\infty n}|^{-(\lambda_1^2-2\lambda_1-1)/(\lambda_1+1)}|A_{\infty n-1}|^{\lambda_1-1}. \tag{40}$$

Combining this result with (27) and (33) proves the theorem. □

Numerical illustrations of Theorem 1 are given in Appendix B.

5. *Alternatives*

A disadvantage of our algorithm is that $x_{p,1}$ in iteration p is calculated differently from the $x_{p,i}$ for $i > 1$. We did so to calculate $x_{p,1}$ as the most accurate approximant possible from the set $\{x_{p-1,1},\ldots,x_{p-1,n+2}\}$. If it is preferred to calculate $x_{p,i}$ in the same way on all processors i (for example, to enable parallelization on a SIMD architecture) one can calculate $x_{p,1}$ as $a_{n-1}(x_{p-1,2},\ldots,x_{p-1,n+2})$. A partial calculation suggests that, as a result, the order of convergence drops to exactly $n+1$ when using $n+2$ processors. This corresponds to an order $\phi = 2$ for the 'three-core secant method' of Corliss.

One can potentially increase the order of convergence by calculating $x_{p,1}$ as the most accurate approximant we can obtain with inverse interpolation and $x_{p,2}$ as the most accurate approximant we obtain with approved approximants. Call a_m^{IP} the approximant calculated with inverse polynomial interpolation and a_m^{IA} the approximant calculated with the method of improved approximants. Then $x_{p,1} = a_n^{\mathrm{IP}}(x_{p-1,1},\ldots,x_{p-1,n+2})$, $x_{p,2} = a_n^{\mathrm{IA}}(x_{p-1,1},\ldots,x_{p-1,n+2})$, while the remaining $x_{p,i}$ for $i = 3,\ldots,n+2$ are calculated as a_{n-1}

https://doi.org/10.1112/S1461157015000236 Published online by Cambridge University Press

by either inverse interpolation or by improved approximants, as described in § 3. A partial calculation indicates that the order of convergence increases to $(n+1+\sqrt{(n+1)^2+8})/2$ when using $n+2$ processors. In the case of three processor cores, it would mean that the order increases from $1+\sqrt{2}\approx 2.41$ (for the algorithm described in § 3) to $1+\sqrt{3}\approx 2.73$ at the expense of a more complicated, and possibly less efficient, algorithm. The lesser efficiency would be caused by a larger calculation time difference between a_n^{IP} and a_n^{IA} than between a_n and a_{n-1} with either inverse interpolation or improved approximants.

So far we have considered algorithms without memory: the approximants at iteration p are calculated from the approximants at iteration $p-1$. Variations with memory can also be considered, for example $x_{p,i} = a_{n+1}(x_{p-1,1},\ldots,x_{p-1,n+2},x_{p-2,i})$. We have not investigated this any further.

6. *A parallel implementation*

So far, our numerical results were obtained with a code that simulates the presence of multiple cores. This allowed us to demonstrate the convergence properties of the algorithm with high precision. We have also implemented the algorithm in a truly parallel Java program. It codes for the case of three cores. The three function evaluations are carried out on three separate threads in each iteration. Together with the main thread, the program requires four threads. The calculation of the approximants is carried out in the main thread. Because this calculation turned out to be fast (a few milliseconds at most), we felt no need to parallelize it.

Table 2 shows the same information as Table 1 but with the secant algorithm and the parallel algorithm implemented in Java. Round-off errors cause the approximation of α to become equal to α at the fifth iteration.

TABLE 2. *As Table 1 but with the secant algorithm and the parallel algorithm implemented in Java. The figures in blue are affected by round-off errors. The correct figures are shown in Table 1.*

$$f(x)=\frac{x(x^2+x-1)}{x+1}$$

	Secant algorithm	Parallel algorithm		
p	x_p	$x_{p,1}$	$x_{p,2}$	$x_{p,3}$
−1	−0.1			
0	0.1	−0.1	0.1	0.2
1	1.99×10^{-2}	-9.33×10^{-3}	4.66×10^{-2}	1.99×10^{-2}
2	-4.88×10^{-3}	-2.87×10^{-5}	3.77×10^{-4}	9.22×10^{-4}
3	1.99×10^{-4}	-3.00×10^{-11}	5.30×10^{-8}	2.17×10^{-8}
4	1.92×10^{-6}	-8.54×10^{-26}	1.30×10^{-18}	3.18×10^{-18}
5	-7.65×10^{-10}	0	0	0

TABLE 3. *The average time t_s of one iteration in the sequential algorithm and, likewise, t_p for the parallel algorithm, along with the parallelization overhead $\delta t = t_p - t_s$. The calculation of each function value $f(x)$ was paused by an amount of time called the sleep time. The function f is the same as in Table 1. All times are in milliseconds. All times are obtained on an Intel Core2 Quad Q6600 @ 2.4 GHz.*

Sleep time	0	20	50	100	200	500
t_s	4.3	22.9	52.8	103.0	203.0	502.8
t_p	12.4	29.5	59.2	109.9	209.7	510.5
δt	8.1	6.6	6.5	6.9	6.8	7.6

https://doi.org/10.1112/S1461157015000236 Published online by Cambridge University Press

The usefulness of a parallel algorithm is determined by two opposing effects: the algorithm is expected to require fewer iterations (because it has a larger order of convergence) but each iteration is slower because of the parallelization overhead. If we use a simple model and estimate the total execution time of the sequential algorithm as the number of iterations N_s times the average time per iteration t_s, (and, likewise, the total execution time for the parallel algorithm as $N_p t_p$), and if the parallel algorithm requires ΔN fewer iterations ($N_p = N_s - \Delta N$) which, however, last δt longer ($t_p = t_s + \delta t$), then the condition $N_p t_p < N_s t_s$ is equivalent to

$$\frac{\Delta N}{N_s} > \frac{\delta t}{t_s + \delta t}. \tag{41}$$

The number of iterations N_s depends on, for example, the nature of the function f, the initial estimates, the sequential algorithm that is used and the required precision with which the root α must be calculated. The reduction in the number of iterations ΔN also depends on these factors as well as on the number of processor cores employed. The average iteration time t_s depends (for a given hardware configuration) on f and the sequential algorithm one is using. All these factors make it hard to make general and, at the same time, practical statements on when to consider a parallel algorithm. As a rule of the thumb though, we expect parallelization to make a difference if either the calculation of f is significantly slower than the expected parallelization overhead δt, or the required precision of the approximation of α is high enough that ΔN becomes of the order of N_s.

We estimate the parallelization overhead δt for our three-core parallel Java program. We do this on an Intel Core2 Quad Q6600 CPU running at 2.4 GHz. This CPU has four processor cores. We estimate the average iteration time t_s of a sequential algorithm which calculates the second-order approximant a_2 with the method of improved approximants. As for the parallel algorithm, the sequential algorithm is implemented in Java. We estimate t_p from the parallel program and calculate $\delta t = t_p - t_s$.

To study the effect of a varying t_p, we do the following. We either pause the threads on which f is calculated (this is the main thread in case of the sequential algorithm) for a certain 'sleep time', or we invoke a loop in the calculation of f in which dummy calculations are carried out. Controlling the number of loop iterations allows us to vary the calculation time of f. The results are shown in Tables 3 and 4.

Table 3 shows a constant parallelization overhead of $7 \sim 8$ ms over a wide range of sleep times. The same overhead occurs in Table 4 for the smallest values of t_s. However, when it takes more time to calculate f, the parallelization overhead increases. We suspect that this is due to the fact that there is more variation in calculation time per core, compared to Table 3. The cores have to perform actual calculations now, rather than being paused for a specified amount of time. Since the time of an iteration is determined by the slowest core, and the variation in calculation time over the cores increases with the time needed to calculate f, the parallelization overhead increases with t_p. This is the more realistic case, compared to pausing the threads.

TABLE 4. *As Table 3 but now we vary the calculation time of f by the invocation of a loop with dummy calculations. The loop length is the number of iterations in the loop in millions.*

Loop length	0	1	2	4	8	16	$\times 10^6$
t_s	2.8	29.0	54.9	106.2	209.8	418.0	
t_p	9.2	35.8	62.6	116.4	225.9	443.1	
δt	6.4	6.8	7.7	10.2	16.1	25.2	

https://doi.org/10.1112/S1461157015000236 Published online by Cambridge University Press

We conclude that our three-core Java program running on our hardware, has a parallelization overhead of at most roughly 10% of the iteration time of the sequential algorithm, with a minimum of about 8 ms. This brings the right-hand side of (41) to about 0.1 if the calculation time of $f(x)$ is much larger than 8 ms.

7. *Conclusions*

We have constructed a parallel root-finding algorithm and calculated the order with which the most accurate approximant converges to the root. We also calculated the corresponding asymptotic error. Our algorithm is for a real, scalar root. It is derivative free and performs one function evaluation per core in each iteration. We have demonstrated the mathematical properties with high-precision numerical simulations of an n-core algorithm. A truly parallel Java implementation showed a parallelization overhead of at most 10% with a minimum of about 8 ms per iteration for the three core algorithm running on a quad-core Intel processor.

The algorithm requires at least three processor cores. The order grows linearly with the number of processors n and becomes approximately equal to $n-1$ when n becomes large. We sketched some alternative algorithms. The versions without memory all appear to have an order of convergence $n-1+\mathcal{O}(1/n)$ for large n. One might hope that the leading order is larger for algorithms with memory. However, this remains to be shown.

We only studied the convergence of the most accurate approximant, that is, of the sequence $\{x_{p,1}\}$. It can be expected that the other sequences $\{x_{p,i}\}$ with $i>1$ converge with the same order, but with a different asymptotic error.

It would be worthwhile to also have a parallelization scheme for two cores. Starting from a sequential algorithm running on one processor core, the first step in parallelization is to run on two cores. A two-core algorithm could be a scheme with memory like $x_{p,1} = a_1(x_{p-1,1}x_{p-1,2}, x_{p-2,1})$ and $x_{p,2} = a_1(x_{p-1,1}x_{p-1,2}, x_{p-2,2})$. We have not explored this any further.

Executing $n+2$ iterations of a sequential algorithm a_m such as discussed in § 2, is expected to give a better approximant of α than one iteration of our parallel algorithm on $n+2$ cores. If one has to solve N independent equations $f_i(x)=0$ with N larger than or equal to the number of available cores, the likely fastest strategy is therefore to solve each equation with a sequential algorithm on one core. Judged by the orders of convergence, this is more efficient than solving the equations one-by-one with a parallel algorithm on all available cores.

Popular 'general purpose' root-finding algorithms for scalar, real roots combine an interval method with a locally convergent method [**2**, **4**, **18**]. Since we know how to parallelize an interval method and we have parallel schemes for locally convergent methods, it should be possible to create a parallel version of these combined methods.

Appendix A. *Recursion with disturbance*

We show that the recursion

$$\begin{aligned} R_n &= \tau_n R_{n-1}^{-\beta}, \quad n = 1, 2, 3, \ldots, \\ R_0 &= 1, \end{aligned} \tag{A.1}$$

with $\tau_n > 0$ for all n, $\lim_{n\to\infty}\tau_n = 1$, and $0<\beta<1$, generates a sequence $\{R_n\}$ which converges to one. This is trivial if $\tau_n = 1$ for all n since then $R_n = R_0^{(-\beta)^n}$. The question is whether the 'disturbances' τ_n can change this limiting behaviour. It may be clear that is not the case, but we give a formal proof below.

https://doi.org/10.1112/S1461157015000236 Published online by Cambridge University Press

Proof. With

$$b_n = (\tau_{n+2}\tau_{n+1}^{-\beta})^{1/(1-\beta^2)},$$

we have

$$R_{n+2} = b_n^{1-\beta^2} R_n^{\beta^2}.$$

We prove that the subsequence $\{R_{2n}\}_{n=0}^{\infty}$ converges to one. From (A.1) it follows that the subsequence $\{R_{2n+1}\}_{n=0}^{\infty}$ also converges to one, which means that the entire sequence $\{R_n\}_{n=0}^{\infty}$ converges to one.

Define $S_n = R_{2n}$ and $c_n = b_{2n}$. S_n develops recursively as

$$S_{n+1} = c_n^{1-\beta^2} S_n^{\beta^2}.$$

We have to prove that $\lim_{n\to\infty} S_n = 1$. From the recursion we see that

$$\begin{aligned} &S_n < S_{n+1} < c_n \quad \text{if } S_n < c_n, \\ &c_n < S_{n+1} < S_n \quad \text{if } S_n > c_n, \\ &S_{n+1} = S_n \qquad\quad \text{if } S_n = c_n, \end{aligned} \tag{A.2}$$

and therefore

$$|S_{n+1} - 1| \leqslant \max(|S_n - 1|, |c_n - 1|). \tag{A.3}$$

Because $\lim_{n\to\infty} c_n = 1$, there exists an $N_\epsilon \in \mathbb{N}$ for all $\epsilon > 0$ such that $|c_n - 1| < \epsilon$ for all $n > N_\epsilon$. If we can proof that for each $\epsilon > 0$ there exists at least one $n^\star > N_\epsilon$ such that $|S_{n^\star} - 1| < \epsilon$, then, from (A.3), $|S_{n^\star+1} - 1| < \epsilon$ and, by induction, $|S_{n^\star+p} - 1| < \epsilon$ for all $p = 0, 1, 2, \ldots$. This implies that the sequence $\{S_n\}$ converges to one.

It remains to be proven that for each $\epsilon > 0$ there exists an $n^\star > N_\epsilon$ such that $|S_{n^\star} - 1| < \epsilon$. Suppose this is not the case. Then there exists an $\tilde{\epsilon}$ and an $\tilde{N}$ such that $|S_n - 1| > \tilde{\epsilon}$ for all $n \geqslant \tilde{N}$. We can take $\tilde{N} > N_{\tilde{\epsilon}}$ such that $|c_n - 1| < \tilde{\epsilon}$ for all $n \geqslant \tilde{N}$. From (A.2), we see that the sequence $\{S_{\tilde{N}+p}\}_{p=0}^{\infty}$ is either decreasing and bounded from below by $1 + \tilde{\epsilon}$, or increasing and bounded from above by $1 - \tilde{\epsilon}$. This means it converges to a limit a. The limit a must be larger than zero. From

$$\lim_{n\to\infty} \frac{S_{n+1}}{S_n^{\beta^2}} = \lim_{n\to\infty} c_n^{1-\beta^2} = 1$$

we have $a = 1$. This contradicts our assumption that there exists an $\tilde{\epsilon}$ and an $\tilde{N}$ such that $|S_n - 1| > \tilde{\epsilon}$ for all $n \geqslant \tilde{N}$. Hence we have proven that for each N_ϵ there exists at least one $n^\star > N_\epsilon$ such that $|S_{n^\star} - 1| < \epsilon$. □

Appendix B. *Numerical examples*

Tables B1 and B2 show numerical examples of the convergence behaviour of theorem 1. Both tables use the function f that is also used in Table 1. It is shown in appendix C of [**17**] that $C_2 = -2$ and $C_n = (-1)^{n+1}$ for $n > 2$ for this function. This gives $A_{\infty 0}^{\mathrm{IP}} = -2$, $A_{\infty 1}^{\mathrm{IP}} = 7$, $A_{\infty 2}^{\mathrm{IP}} = -31$ and $A_{\infty 3}^{\mathrm{IP}} = 154$ for the method of inverse polynomial interpolation. We have $A_{\infty 0}^{\mathrm{IA}} = -2$, $A_{\infty 1}^{\mathrm{IA}} = 3$, $A_{\infty 2}^{\mathrm{IA}} = -5$ and $A_{\infty 3}^{\mathrm{IA}} = 8$ for the method of improved approximants.

The tables below were calculated using high-precision arithmetic.

https://doi.org/10.1112/S1461157015000236 Published online by Cambridge University Press

TABLE B1. *Convergence of the three-core parallel algorithm for the function f. Convergence is shown for both the method of inverse polynomial interpolation and the method of improved approximants. Starting values are $x_{0,1} = -0.1$, $x_{0,2} = 0.1$ and $x_{0,3} = 0.2$ for both methods. The order of convergence is $\phi_3 = 1 + \sqrt{2}$ (c.f. Theorem 1). The expected value is the asymptotic error term $|A_{\infty 0}|^{\phi_3 - 1}$ (c.f. Theorem 1). The expected values are the same for both methods because $A^{IP}_{\infty 0} = A^{IA}_{\infty 0} = C_2$. The first 5 iterations for the method of improved approximants are also shown in Table 1.*

$$f(x) = \frac{x(x^2 + x - 1)}{x + 1}$$

	Number of cores: 3			
	Inverse interpolation		Improved approximants	
p	$x_{p,1}$	$\dfrac{\lvert x_{p,1}\rvert}{\lvert x_{p-1,1}\rvert^{\phi_3}}$	$x_{p,1}$	$\dfrac{\lvert x_{p,1}\rvert}{\lvert x_{p-1,1}\rvert^{\phi_3}}$
1	-2.74×10^{-2}	7.105 39	-9.33×10^{-3}	2.422 025
2	-1.97×10^{-4}	1.165 54	-2.87×10^{-5}	2.283 310
3	-3.93×10^{-9}	3.481 66	-3.00×10^{-11}	2.776 358
4	-1.21×10^{-20}	2.377 37	-1.03×10^{-25}	2.619 726
5	-2.32×10^{-48}	2.794 31	-1.28×10^{-60}	2.684 187
6	-2.60×10^{-115}	2.613 41	-6.77×10^{-145}	2.657 296
7	-6.29×10^{-277}	2.686 87	-2.35×10^{-348}	2.668 402
8	-4.12×10^{-667}	2.656 19	-1.49×10^{-839}	2.663 796
9	-4.27×10^{-1609}	2.668 86	-2.09×10^{-2025}	2.665 703
10	-3.00×10^{-3883}	2.663 61	-2.62×10^{-4888}	2.664 913
11	-1.53×10^{-9373}	2.665 78	$-5.74 \times 10^{-11\,800}$	2.665 240
12	$-2.82 \times 10^{-22\,628}$	2.664 88	$-3.44 \times 10^{-28\,486}$	2.665 104
13	$-4.89 \times 10^{-54\,628}$	2.665 25	$-2.72 \times 10^{-68\,770}$	2.665 161
14	$-2.71 \times 10^{-131\,882}$	2.665 10	$-1.02 \times 10^{-166\,024}$	2.665 137
15	$-1.43 \times 10^{-318\,390}$	2.665 16	$-1.13 \times 10^{-400\,817}$	2.665 147
16	$-2.23 \times 10^{-768\,661}$	2.665 14	$-5.23 \times 10^{-967\,658}$	2.665 144
Expected		2.66 514		2.665 144

TABLE B2. *As table B1 but for four and five cores. The method of inverse polynomial interpolation is used for four cores, the method of improved approximants for five cores. Starting values are $x_{0,1} = -0.2$, $x_{0,2} = -0.1$, $x_{0,3} = 0.1$ and $x_{0,4} = 0.2$ in the case of four cores. The order of convergence is $\phi_4 = (3 + \sqrt{13})/2$ and the asymptotic error is $|A_{\infty 2}|^{-\phi_4/(\phi_4+1)}|A_{\infty 1}|^{\phi_4 - 1}$. Starting values are $x_{0,1} = -0.2$, $x_{0,2} = -0.1$, $x_{0,3} = 0.1$, $x_{0,4} = 0.2$ and $x_{0,5} = 0.3$ in the case of five cores. The order of convergence is $\phi_5 = 2 + \sqrt{5}$ and the asymptotic error is $|A_{\infty 3}|^{-2\phi_5/(\phi_5+1)}|A_{\infty 2}|^{\phi_5 - 1}$.*

$$f(x) = \frac{x(x^2 + x - 1)}{x + 1}$$

	Number of cores: 4		Number of cores: 5	
	Inverse interpolation		Improved approximants	
p	$x_{p,1}$	$\dfrac{\lvert x_{p,1}\rvert}{\lvert x_{p-1,1}\rvert^{\phi_4}}$	$x_{p,1}$	$\dfrac{\lvert x_{p,1}\rvert}{\lvert x_{p-1,1}\rvert^{\phi_5}}$
1	-1.62×10^{-2}	3.2965	2.14×10^{-3}	1.952 46
2	-4.50×10^{-6}	3.6920	-3.06×10^{-11}	6.265 69
3	-1.62×10^{-17}	7.3504	1.83×10^{-44}	6.354 45
4	-2.11×10^{-55}	6.0475	-3.35×10^{-185}	6.311 30
5	-1.67×10^{-180}	6.4164	2.25×10^{-781}	6.321 46
6	-1.09×10^{-593}	6.3024	-8.34×10^{-3307}	6.319 06
7	-2.36×10^{-1958}	6.3367	$1.06 \times 10^{-14\,004}$	6.319 63
8	-1.59×10^{-6465}	6.3263	$-1.04 \times 10^{-59\,321}$	6.319 50
9	$-1.04 \times 10^{-21\,351}$	6.3294	$1.21 \times 10^{-251\,287}$	6.319 53
10	$-1.99 \times 10^{-70\,517}$	6.3285		
11	$-9.12 \times 10^{-232\,901}$	6.3288		
12	$-1.67 \times 10^{-769\,216}$	6.3287		
Expected		6.3287		6.319 52

https://doi.org/10.1112/S1461157015000236 Published online by Cambridge University Press

Acknowledgements. The author thanks Professor George F. Corliss for an explanation of his work on parallel root-finding algorithms. Dr A. Hogervorst has created and run the parallel Java program. I have benefited from many discussions with him about the results of § 6. The high-precision numerical results have been obtained with the help of the open-source program Sage (www.sagemath.org). The Java programs have been created in the open-source development environment NetBeans (www.netbeans.org).

References

1. G. E. Alefeld and F. A. Potra, 'Some efficient methods for enclosing simple zeros of nonlinear equations', *BIT* 32 (1992) no. 2, 334–344.
2. G. E. Alefeld, F. A. Potra and Y. Shi, 'Algorithm 748: enclosing zeros of continuous functions', *ACM Trans. Math. Software* 21 (1995) no. 3, 327–344.
3. S. Amat and S. Busquier, 'On a higher order secant method', *Appl. Math. Comput.* 141 (2003) no. 2–3, 321–329.
4. R. P. Brent, 'An algorithm with guaranteed convergence for finding a zero of a function', *Comput. J.* 14 (1971) no. 4, 422–425.
5. R. Brent, S. Winograd and P. Wolfe, 'Optimal iterative processes for root-finding', *Numer. Math.* 20 (1973.) 327–341.
6. R. L. Burden, J. D. Faires and A. C. Reynolds, *Numerical analysis*, 2nd edn (Prindle, Weber and Schmidt, 1981).
7. A. Cordero, J. L. Hueso, E. Martínez and J. R. Torregrosa, 'A family of derivative-free methods with high order of convergence and its application to nonsmooth equations', *Abstr. Appl. Anal.* 2012 (2012) Article ID 836901, doi:10.1155/2012/836901.
8. G. F. Corliss, 'Parallel root finding algorithms', PhD Thesis, Department of Mathematics, Michigan State University, 1974.
9. P. Díez, 'A note on the convergence of the secant method for simple and multiple roots', *Appl. Math. Lett.* 16 (2003) no. 8, 1211–1215.
10. S. Gal and W. Miranker, 'Optimal sequential and parallel search for finding a root', *J. Combin. Theory Ser.* A 23 (1977) 1–14.
11. K. Ghidouche, R. Couturier and A. Sider, 'A parallel implementation of the Durand–Kerner algorithm for polynomial root-finding on GPU', *2014 International Conference on Advanced Networking Distributed Systems and Applications (INDS)* (IEEE, 2014) 53–57.
12. F. B. Hildebrand, *Introduction to numerical analysis*, 2nd edn (Dover, 1987).
13. B. Ignatova, N. Kyurkchiev and A. Iliev, 'Multipoint algorithms arising from optimal in the sense of Kung–Traub iterative procedures for numerical solution of nonlinear equations', *Gen. Math. Notes* 6 (2011) no. 2, 45–79.
14. D. E. Muller, 'A method for solving algebraic equations using an automatic computer', *Mathematical Tables and Other Aids to Computation* 10 (1956) 208–215.
15. R. E. Moore, R. B. Kearfott and M. J. Cloud, *Introduction to interval analysis* (SIAM, 2009).
16. H. Mukai, 'Parallel algorithms for solving systems of nonlinear equations', *Comput. Math. Appl.* 7 (1981) 235–250.
17. M. J. P. Nijmeijer, 'A method to accelerate the convergence of the secant algorithm', *Adv. Numer. Anal.* 2014 (2014) Article ID 321592, doi:10.1155/2014/321592.
18. E. Novak, K. Ritter and H. Wozniakowski, 'Average case optimality of a hybrid secant-bisection method', *Math. Comp.* 64 (1995) no. 212, 1517–1539.
19. V. Y. Pan and A.-L. Zheng, 'New progress in real and complex polynomial root-finding', *Comput. Math. Appl.* 61 (2011) no. 5, 1305–1334.
20. W. H. Press, S. A. Teukolsky, W. T. Vetterling and B. P. Flannery, *Numerical recipes: the art of scientific computing*, 2nd edn (Cambridge University Press, 2007).
21. M. Raydan, 'Exact order of convergence of the Secant method', *J. Optim. Theory Appl.* 78 (1993) 541–551.
22. J. R. Sharma and P. Gupta, 'An efficient family of Traub–Steffensen-type methods for solving systems of nonlinear equations', *Adv. Numer. Anal.* 2014 (2014) Article ID 152187, doi:10.1155/2014/152187.
23. A. Sidi, 'Unified Treatment of Regula Falsi, Newton–Raphson, Secant, and Steffensen Methods for Nonlinear Equations', *J. Online Math. Appl.* 6 (2006), www.maa.org/node/115943.
24. A. Sidi, 'Generalization of the Secant method for nonlinear equations', *Appl. Math. E-Notes* 8 (2008) 115–123.
25. J. Stoer and R. Bulirsch, *Introduction to numerical analysis*, 3rd edn (Springer, 2002).
26. L. Taher and T. Elahe, 'On a new efficient Steffensen-like iterative class by applying a suitable self-accelerator parameter', *Sci. World J.* 2014 (2014) doi:10.1155/2014/769758.

27. J. Traub, *Iterative methods for the solution of equations* (Prentice-Hall, Englewood Cliffs, New Jersey, 1964.).
28. H. T. Kung and J. F. Traub, 'Optimal order of one-point and multipoint iteration', *J. Assoc. Comput. Mach.* 21 (1974) no. 4, 643–651.
29. H. T. Kung and J. F. Traub, 'Optimal order and efficiency for iterations with two evaluations', *SIAM J. Numer. Anal.* 13 (1976) no. 1, 84–99.

M. J. P. Nijmeijer
Heemraadssingel 182D
3021 DM Rotterdam
The Netherlands
mail@marconijmeijer.nl

https://doi.org/10.1112/S1461157015000236 Published online by Cambridge University Press