Computers and Mathematics with Applications 63 (2012) 239–254

Contents lists available at SciVerse ScienceDirect

Computers and Mathematics with Applications

journal homepage: www.elsevier.com/locate/camwa

Efficient polynomial root-refiners: A survey and new record efficiency estimates

J.M. McNamee [a], Victor Y. Pan [b,c,*]

[a] Department of Computer Science and Engineering, York University, 4700 Keele St, Toronto, ON, M3J 1P3, Canada

[b] Department of Mathematics and Computer Science, Lehman College of the City University of New York, Bronx, NY 10468, USA

[c] Ph.D. Programs in Mathematics and Computer Science, The Graduate Center of the City University of New York, New York, NY 10036, USA

ARTICLE INFO

Article history:
Received 7 November 2011
Accepted 9 November 2011

Keywords:
Iterative root-refiners
Efficiency
Simultaneous iterations
Polynomial factorization
Companion matrix methods

ABSTRACT

A typical iterative polynomial root-finder begins with a relatively slow process of computing a crude but sufficiently close initial approximation to a root and then rapidly refines it. The policy of using the same iterative process at both stages of computing an initial approximation and refining it, however, is neither necessary nor most effective. The efficiency of an iteration at the former stage resists formal study and is usually decided empirically, whereas formal study of the efficiency at the latter stage of refinement is not hard and is the subject of the current paper. We define this local efficiency as $\frac{\log_{10} q}{d} = \log_{10}(q^{1/d})$ (q is the convergence order, and d is the number of function evaluations per iteration); it is inversely proportional to the number of flops involved. Assuming that about $2n$ flops are needed per evaluation of a polynomial of a degree n at a single point, we extend the definition to cover the recent matrix methods for polynomial root-finding as well as some methods that combine n approximations to all n roots to refine them simultaneously. For the approximation of a single root of a polynomial of degree n, the maximum local efficiency achieved so far is $\log_{10} 2 \approx 0.301\ldots$, but we show its growth to infinity for simultaneous approximation of all n roots as n grows to infinity.

© 2011 Elsevier Ltd. All rights reserved.

1. Introduction

1.1. Polynomial root-finding: some history

Univariate polynomial root-finding is the oldest problem of mathematics and computational mathematics; it was central for these fields from the Sumerian and Babylonian times (the third millennium B.C.) and well into the XIXth century A.D. Its study was responsible for the introduction of some basic concepts and algorithms, e.g., groups, fields, algebraic and meromorphic functions, the Regula Falsi and secant algorithms, and irrational, negative, and nonreal numbers. For more on the rich and exciting history of this study see, for example, the books [1,2] and surveys [3,4].

At present, univariate polynomial root-finding is a fundamental operation of computer algebra, particularly critical for geometric modelling and algebraic geometric computations such as the computation of the intersection of algebraic curves and surfaces, which amounts to the solution of systems of polynomial equations. The most popular solution algorithms, employing Gröbner bases, reduce the latter task to accurate root-finding for high degree univariate polynomials.

* Corresponding author at: Department of Mathematics and Computer Science, Lehman College of the City University of New York, Bronx, NY 10468, USA. Tel.: +1 914 737 2637; fax: +1 718 960 8969.

E-mail addresses: mcnamee@cse.yorku.ca (J.M. McNamee), victor.pan@lehman.cuny.edu, v_y_pan@yahoo.com (V.Y. Pan).

URL: http://comet.lehman.cuny.edu/vpan/ (V.Y. Pan).

0898-1221/$ – see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.camwa.2011.11.015

Other important application areas of polynomial root-finding include control and signal processing (see the demonstrations on pages XIV and XV of [5]).

Over the centuries, especially since the advent of computers, there have been hundreds if not thousands of different methods devised for numerical approximation of the roots of polynomials. A large number (perhaps most) of them are listed in the bibliography by McNamee [6], and many are described in the two-part monograph by McNamee and Pan [7,5].

1.2. Our subject: iterative techniques for polynomial root-finding and root-refining

The roots of a polynomial of a degree higher than four must be computed approximately because there exist no explicit expressions for them using arithmetic operations and radicals, and so the known methods are iterative; for example, they begin with a sufficiently close initial approximation x_0 to a root and recursively improve it.

Simultaneous methods proceed similarly for all roots of a polynomial (see Section 5), whereas the matrix root-finders use iterative eigenvalue algorithms applied to companion or generalized companion matrices associated with a given polynomial (see Section 4).

A typical iterative polynomial root-finder begins with a relatively slow process of computing a crude but reasonably close initial approximation to a root and then rapidly refines it. The policy of using the same iterative process at both stages of computing an initial approximation and refining it, however, is neither necessary nor most effective. The efficiency of an iteration at the former stage resists formal study and is usually decided empirically (see [8]), although this difficulty is overcome, for example, in [9–13]. In contrast, estimating the efficiency of the latter stage of refinement is not so hard, and is the subject of the current paper.

Namely, we assume that some black box algorithms have already supplied desired initial approximations to the roots; then we study *root-refining* by iterations that converge with order q right from the start. For example, $q = 2$ for Newton's method provided that a root lies substantially closer to an initial approximation than to any other root. Reneger [14] provides specific estimates based on the results of [15,16], and this study has been extended to other root-finders as well (see [17,18]).

1.3. Iterative root-finders and root-refiners: classification

We assume a polynomial equation $f(x) = 0$, and a reasonably good initial approximation x_0 to its root $x = \zeta$. Then we apply an iterative method such as $x_{i+1} = \phi(\cdots)$, where ϕ depends for example on $x_i, f(x_i)$ and its derivatives, repeating the iteration until some convergence criterion is satisfied. Or the iteration may be more complicated (see (2)–(4) below). The following classes cover a large part of the iterative methods of Sections 2 and 3; in particular, classes (3) and (4) are covered in Sections 2.2 and 3.2.

(1) *One point, without memory*: the new approximation x_{i+1} is defined as a function of

$$x_i, f(x_i), f'(x_i), \ldots, f^{(s-1)}(x_i) \quad \text{for } i = 0, 1, \ldots, \text{ i.e.,}$$
$$x_{i+1} = \phi(x_i, f(x_i), f'(x_i), \ldots, f^{(s-1)}(x_i)).$$

(2) *One point, with memory*. Here we re-use old values:

$x_{i-1}, \ldots, x_{i-m}$ for a fixed positive integer m, i.e., x_{i+1} is given by

$$\phi(x_i, x_{i-1}, \ldots, x_{i-m}, f(x_{i-1}), \ldots, f(x_{i-m}), \ldots, f^{(s-1)}(x_{i-1}), \ldots, f^{(s-1)}(x_{i-m})).$$

The overall work (over all $i = 1, 2, \ldots, k$) is almost the same as for one point without memory provided that k is large enough.

(3) *Multi-point (without memory)*. It proceeds thus:

$$z_1 = \phi_1(x_i, f(x_i), \ldots, f^{(s-1)}(x_i)),$$
$$z_2 = \phi_2(x_i, f(x_i), \ldots, f^{(s-1)}(x_i), z_1, f(z_1), \ldots, f^{(s-1)}(z_1)),$$
$$\ldots\ldots\ldots\ldots$$
$$\ldots\ldots\ldots\ldots$$
$$z_j = \phi_j(x_i, f(x_i), \ldots, f^{(s-1)}(x_i), z_1, f(z_1), \ldots, f^{(s-1)}(z_1), \ldots,$$
$$z_{j-1}, f(z_{j-1}), \ldots, f^{(s-1)}(z_{j-1})) \quad (j = 3, \ldots, n),$$
$$x_{i+1} = z_n,$$

where $n \geq 4$ is a fixed integer.

(4) *Multi-point with memory*: as above, but we also use old values:

$$x_{i-\ell}, f(x_{i-\ell}), \ldots, f^{(s-1)}(x_{i-\ell}) \quad (\ell = 1, \ldots, m).$$

1.4. Efficiency: definition

We define the efficiency of an iterative root-refiner as the inverse of the work needed to obtain a desired accuracy of solution. This may be derived as follows: suppose we start with a guess having error 10^{-1} and seek an estimate within the

error bound 10^{-D}. Let the method have order q, i.e., for a root ξ, we have

$$\frac{|x_{i+1}-\xi|}{|x_i-\xi|^q}=C\neq 0,\infty,$$

where q is not necessarily an integer. For the sake of demonstration, let $C=1$. Then, after one step, the initial error bound $\epsilon_0=10^{-1}$ decreases to $\epsilon_1=10^{-q}$, after two steps, to $\epsilon_2=(10^{-q})^q=10^{-q^2}$, and so on.

Finally, after N steps, the output error is bounded by $\epsilon_N=10^{-q^N}$ and must stay below 10^{-D}. Therefore we should choose

$$N=\left\lceil\frac{\log D}{\log q}\right\rceil\approx\frac{\log D}{\log q}.$$

Hereafter, log stands for $\log_{10}$; that is, we assume logarithm to base 10.

Now, suppose that each iteration requires d function or derivative evaluations (in our crude estimate of efficiency we ignore the cost of combining the different function values; for high degrees this makes little difference). Then the work equals $Nd\approx\log D\frac{d}{\log q}$. Hence, the efficiency equals

$$\frac{1}{Work}\approx\frac{1}{\log D}\frac{\log q}{d},$$

but D is problem dependent rather than method dependent, and it is usual to ignore it. Thus we finally define

$$Eff=\frac{\log q}{d}=\log\left(q^{\frac{1}{d}}\right). \tag{1.1}$$

Many authors use just $q^{\frac{1}{d}}$, but the back and forth transition to our definition is immediate. For an example, Newton's method has $q=d=2$, and so $q^{\frac{1}{d}}=\sqrt{2}=1.414$, whereas our definition gives $Eff=\log\sqrt{2}=\frac{1}{2}\log 2=.1505$.

The increase of the efficiency in (1.1) by a factor f corresponds to the decrease by the same factor of the arithmetic time of the solution, measured by the number of flops involved.

1.5. Contents

In the next two sections, we recall a number of iterative methods which have efficiencies in the above sense of (somewhat arbitrarily) more than .225; we classify them depending on their derivation and whether they use only the values of the input polynomial or also its derivatives and possibly higher-order derivatives. Table 1 at the end of the paper summarizes the respective estimates. In Sections 4 and 5, we recall that the evaluation of a polynomial of a degree n at a point takes about $2n$ flops, extend our study to cover matrix methods for polynomial root-finding and root-refining (in Section 4) and simultaneous methods (in Section 5), and show that for the class of the latter methods the efficiency can dramatically increase. Section 6 concludes the paper.

2. Iterative methods not using derivatives

2.1. Interpolation methods

We consider first Muller's method [19], derived by fitting a quadratic through the three most recent approximations $(x_j,f(x_j))(j=i,i-1,i-2)$ and finding where the quadratic cuts the x-axis. By applying Newton's divided difference interpolation formula, we compute

$$x_{i+1}=x_i-\frac{2f(x_i)}{b\pm\sqrt{b^2-4af(x_i)}},$$

where

$$a=[x_i,x_{i-1},x_{i-2}]=\frac{[x_i,x_{i-1}]-[x_{i-1},x_{i-2}]}{x_i-x_{i-2}},$$
$$b=[x_i,x_{i-1}]+(x_i-x_{i-1})a,$$
$$[x_i,x_{i-1}]=\frac{f(x_i)-f(x_{i-1})}{x_i-x_{i-1}},$$

and $[x_{i-1},x_{i-2}]$ is similar. If the quantity under the square root sign is positive, we take the sign of the square root to be that of b; otherwise, we take the sign which will make the denominator larger in magnitude. Householder [20] shows that the order is 1.839, and since we only need one new function evaluation per step, the efficiency is $\log 1.839=.265$. For a Fortran program, see [21].

Sharma [22] generalizes Muller's method to a family of methods based on fitting the general quadratic

$$Q(x, y) = ax^2 + by^2 + cx + dy + e$$

through the three points $(x_j, f_j)(j = i, i-1, i-2)$. He thus obtains the iteration

$$x_{i+1} = x_i - \frac{2f_i(bf_i - d)}{c \pm \sqrt{c^2 - 4af_i(bf_i - d)}}.$$

Here, c and d depend on a, b, and the (x_j, f_j), while a and b may be chosen to give different members of the family. For example, $a = 0, b = 1$ gives inverse parabolic interpolation (see the cited paper for details). The order and efficiency are the same as for Muller's method.

Jarratt [23] considers general polynomial interpolation (direct and inverse). The latter case, which is easier, proceeds by fitting

$$x = \sum_{j=0}^{n} a_j y^j$$

to the points $(f_{i-k}, x_{i-k})(k = 0, 1, \ldots, n)$ by Lagrange's or Newton's divided difference interpolation. Then the next estimate is given by $y = 0$ or $x_{i+1} = a_0$. For $n = 4$, the order is 1.984, giving an efficiency of .2975.

The secant method uses

$$x_{i+1} = x_i - f_i \frac{x_i - x_{i-1}}{f_i - f_{i-1}}.$$

A variation known as Regula Falsi starts with $f(x_0)f(x_1) < 0$ and at each iteration replaces whichever of the old points has the sign of the function value the same as that of the new point (the old point is replaced by the new one).

The secant and Regula Falsi methods are not particularly efficient, but Anderson and Björck [24] modify Regula Falsi so that, when

$$f_i f_{i+1} > 0,$$

they use

$$x_{i+2} = x_{i+1} - \frac{f_{i+1}}{\overline{f}'_{i+1}}, \tag{2.1}$$

where $\overline{f}'_{i+1}$ is the derivative of the interpolating parabola to $(x_k, f_k)(k = i-1, i, i+1)$ at x_{i+1}, namely

$$\overline{f}'_{i+1} = [x_{i+1}, x_i] + [x_{i+1}, x_{i-1}] - [x_{i-1}, x_i]. \tag{2.2}$$

(If $f_i f_{i+1} < 0$, an unmodified Regula Falsi step is taken.) Now, if x_{i+2} does not lie in the interval $[x_{i-1}, x_{i+1}]$, we take

$$x_{i+2} = x_{i+1} - (x_{i-1} - x_{i+1}) \frac{f_{i+1}}{\frac{1}{2}f_{i-1} - f_{i+1}}.$$

The efficiency varies between .226 and .233.

King [25] improves the Anderson and Björck method as follows (starting with (x_0, f_0) and (x_1, f_1), where $f_0 f_1 < 0$ as in Regula Falsi):

(1) Find x_{i+1} from (x_j, f_j) $(j = i-1, i)$ by a secant step and calculate f_{i+1}. If $f_i f_{i+1} < 0$, set $(x_B, f_B) = (x_i, f_i)$; else, set $(x_B, f_B) = (x_{i-1}, f_{i-1})$.

(2) Perform computations defined by Eqs. (2.1) and (2.2) above to obtain x_{i+2}. If this is not in $[x_{i+1}, x_B]$, obtain x_{i+2} by using

$$x_{i+2} = x_{i+1} - (x_B - x_{i+1}) \frac{f_{i+1}}{\frac{1}{2}f_B - f_{i+1}}.$$

Replace f_B by $\frac{f_B}{2}$. Calculate f_{i+2}.

(3) If $f_{i+1}f_{i+2} < 0$, set $(x_B, f_B) = (x_{i+1}, f_{i+1})$. In any case, replace (x_{i-1}, f_{i-1}), (x_i, f_i), and (x_{i+1}, f_{i+1}) by (x_i, f_i), (x_{i+1}, f_{i+1}), and (x_{i+2}, f_{i+2}), respectively, and return to step 2.

The order is 1.839, as for Muller's method, but King's method has the advantage of guaranteeing convergence for real roots. For complex roots we can apply other methods such as Muller's.

Kogan [26] presents a method that uses the following extension of the secant method:

$$x_{i+1} = x_i - \frac{f(x_i)}{[x_{i-1}, x_i][x_{i-2}, x_{i-1}] - f(x_{i-1})[x_{i-2}, x_i]}.$$

According to Kogan, this method has order 1.84 and hence efficiency .2648.

A nonstationary method is proposed in [27]:

$$x_{i+1} = x_i - \frac{f(x_i)}{[x_{i-1}, x_i] + \sum_{s=2}^{i} [x_{i-s}, \ldots, x_i] \prod_{j=i-s+1}^{i-1} (x_i - x_j)} \quad (i = 1, 2, \ldots),$$

where

$$[x_{k-s}, \ldots, x_k] = \frac{[x_{k-s+1}, \ldots, x_k] - [x_{k-s}, \ldots, x_{k-1}]}{x_k - x_{k-s}} \quad (s = 1, \ldots, k),$$

with

$$[x_k, x_k] = f(x_k).$$

For example,

$$x_2 = x_1 - \frac{f(x_1)}{[x_0, x_1]},$$

$$x_3 = x_2 - \frac{f(x_2)}{[x_1, x_2] + [x_0, x_1, x_2](x_2 - x_1)},$$

$$x_4 = x_3 - \frac{f(x_3)}{[x_2, x_3] + [x_1, x_2, x_3](x_3 - x_2) + [x_0, \ldots, x_3](x_3 - x_2)(x_3 - x_1)}.$$

The convergence order itself converges to 2 as i increases, so the efficiency approaches $\log 2 = .3010$.

2.2. Multi-point methods

Traub [28] shows that one-point iterations without memory, using $s-1$ derivatives, are of order at most s. With memory, the maximum order is $s + 1$.

Kung and Traub [29] describe a multi-point iteration of order 2^{n-1} using n evaluations and conjecture that this is optimal among all m-point iterations using n evaluations and no memory.

They define a set of iterations $\{\psi_j\}$ as follows:

$$\psi_0 = x,$$

$$\psi_1 = x + \beta f(x),$$

. . . . . .

. . . . . .

$$\psi_{j+1} = Q_j(0),$$

where $Q_j(y)$ is the inverse interpolatory polynomial for f at $f(\psi_k)(k = 0, 1, \ldots, j)$. For example,

$$\psi_2 = x_0 - \frac{\beta f(\psi_0) f(\psi_1)}{f(\psi_1) - f(\psi_0)},$$

$$\psi_3 = \psi_2 - \frac{f(\psi_0) f(\psi_1)}{f(\psi_2) - f(\psi_0)} \left[\frac{\psi_1 - \psi_0}{f(\psi_1) - f(\psi_0)} - \frac{\psi_2 - \psi_0}{f(\psi_2) - f(\psi_0)} \right],$$

and the authors give an Algol program for computing ψ_n for $n \geq 4$. They show that the order is 2^{n-1}, and that, among all Hermite interpolatory function iterations using n evaluations, the above order is maximal.

Werschultz [30] considers a class of multi-point methods with memory using Hermite information, i.e., assuming initial approximation x_0 available, we recursively compute x_{i+1} from x_i using

$$f^{(j)}(z_{i-s,\ell}), \quad 0 \leq j \leq r_\ell - 1,\ 1 \leq \ell \leq k,\ 0 \leq s \leq m$$

for fixed integers $k, m, r_1, \ldots, r_k$. The number of new function evaluations per iteration is

$$n = \sum_{\ell=1}^{k} r_\ell,$$

and the memory is m.

$z_{i,\ell+1}$ depends on

$$z_{i,q}, f(z_{i,q}), f^{(j)}(z_{i,q}) \quad (j = 1, \ldots, r_\ell - 1;\ q = 1, \ldots, \ell);$$

$$z_{i-s,q}, f(z_{i-s,q}), f^{(j)}(z_{i-s,q}) \quad (q = 1, \ldots, k;\ s = 1, \ldots, m;\ j = 1, \ldots, r_\ell - 1).$$

If $k = 1$, we have one-point iterations with memory; if $m = 0$, we have multi-point iterations without memory. Werschultz shows that the order of such methods is bounded by 2^n (so the efficiency is bounded by $\log 2$) and that this is nearly attained for $k = n$, $r_1 = \cdots = r_n = 1$ (i.e., no derivatives used) and for a moderately large m. This result is reached by a Hermite interpolatory method.

2.3. Methods based on rational approximation

Jarratt and Nudds [31] use rational approximation, i.e., they fit

$$f(x) = \frac{x - c}{ax + b}$$

through the three points $(x_{i-j}, f(x_{i-j}))(j = 0, 1, 2)$. Then the next approximation to the roots is given by

$$x_{i+1} = x_i + \frac{(x_i - x_{i-1})(x_i - x_{i-2})f_i(f_{i-1} - f_{i-2})}{(x_i - x_{i-1})(f_{i-2} - f_i)f_{i-1} + (x_i - x_{i-2})(f_i - f_{i-1})f_{i-2}},$$

where of course $f_i = f(x_i)$, etc. The order and efficiency are the same as that of Muller's method, but the method of [31] has the advantage of finding real roots without using complex arithmetic (in contrast to Muller).

Larkin [32] uses a rational approximation, i.e., fits the given polynomial $f(x)$ by

$$\tilde{f}(x) = \frac{x - x_{n+1}}{Q_{n-2}(x)}$$

for a fixed integer $n > 2$. He computes

$$w_{jk} = w_{j+1,k-1} + \frac{w_{j+1,k-1} - w_{j,k-1}}{(w_{j,k-1} - x_j)/(w_{j+1,k-1} - x_{j+k}) - 1}$$

for $j = 1, \ldots, n-1$; $k = 2, \ldots, n-1$, and the w_{j1} given by the secant rule

$$w_{j1} = x_{j+1} - \frac{x_{j+1} - x_j}{f_{j+1} - f_j} f_{j+1},$$

where (starting from initial guesses x_1, x_2) the x_j are given by

$$x_{n+1} = w_{1,n-1} \quad (n = 2, 3, \ldots). \tag{2.3}$$

For example, if we go up to $n = 4$, the order of calculation is $f_1, f_2, w_{11}, x_3 = w_{11}, f_3, w_{21}, w_{12}, x_4 = w_{12}, f_4, w_{31}, w_{22}, w_{13}, x_5 = w_{13}$. Larkin proves that if x_1 and x_2 are close enough to a root ζ, then $x_i \to \zeta$ as $i \to \infty$ with convergence order 2.

The later paper Larkin [33] gives a simpler method (we mention the previous method because there exists a program based on it — see later). He defines

$$h(x) = \frac{1}{f(x)}.$$

Then let $h[x_n, x_{n+1}, \ldots, x_{n+k}]$ denote the nth divided difference based on the points $(x_r, h_r)(r = n, n+1, \ldots, n+k)$ with

$$h[x_j] = h_j \equiv h(x_j) = \frac{1}{f(x_j)} \quad (j = 1, 2, \ldots).$$

Starting from two initial guesses x_1, x_2, he computes

$$x_{n+1} = x_n - \frac{h[x_1, \ldots, x_{n-1}]}{h[x_1, \ldots, x_n]}.$$

The divided differences can be computed recursively by

$$h[x_j, \ldots, x_{j+k}] = \frac{h[x_{j+1}, \ldots, x_{j+k}] - h[x_j, \ldots, x_{j+k-1}]}{x_{j+k} - x_j}. \tag{2.4}$$

There is a problem as n gets larger, namely the divided differences become very large, and there is a danger of overflow. Also for large n the number of 'overhead' calculations (i.e., other than function evaluations) increases. These problems can be overcome if we restrict k in (2.4) to a fixed number such as 5; that is, if we generate $\{x_n\}(n = 1, 2, \ldots, k+1)$ by (2.4), and subsequently use

$$x_{n+1} = x_n + \frac{h[x_{n-k}, x_{n-k+1}, \ldots, x_{n-1}]}{h[x_{n-k}, \ldots, x_n]}.$$

Thus we only use divided differences as far as the k is restricted, and points prior to (x_{n-k}, h_{n-k}) are not used in the calculation of x_{n+1}. The orders of convergence for various values of k are as follows:

k	1	2	$\cdots$	5	$\cdots$	11
q	1.618	1.889	$\cdots$	1.984	$\cdots$	2.000.

Norton [34] describes an algorithm based on Larkin's paper [32] and its combination with bisection where necessary. Along with other parameters, the user needs to set n equal to the maximum degree of rational interpolation. Norton recommends $n = 5$ for simple roots, or $n = 3$ for multiple roots. He suggests a higher value for functions which are hard to evaluate, such as high-degree polynomials.

2.4. Miscellaneous methods

Henrici [35] gives a derivation of Aitken's Δ^2 method (proposed in [36]) for accelerating the convergence of a sequence which converges linearly, such as

$$x_{i+1} = \phi(x_i) \tag{2.5}$$

(the method of successive approximation). The Δ^2 method can be written as

$$x_i' = x_i - \frac{(x_{i+1} - x_i)^2}{x_{i+2} - 2x_{i+1} + x_i} \tag{2.6}$$

(note that the denominator in the right-hand side is $\Delta^2 x_i$; hence the name of the method). It is suggested that we start with x_0, form x_1 and x_2 by (2.5), and apply (2.6) with $i = 0$ to produce x_0'; then repeat the above process to convergence (hopefully). This process has quadratic convergence and evaluates as many functions per iteration as the basic process (2.5). In particular, the efficiency is $\log 2 = .3010$ if a single evaluation is used in (2.5).

Kalantari [37] defines

$$f_i = f(x_i), f_{ij} = \frac{f_{i+1,j} - f_{i,j-1}}{x_j - x_i} \ (x_i \neq x_j),$$

and recommends the iteration in which the 4-tuple (x_1, x_2, x_3, x_4) is replaced by the 4-tuple $(B_4^{(4)}, x_1, x_2, x_3)$ for

$$B_4^{(4)} = x_1 - \frac{f_{11}\begin{vmatrix} f_{23} & f_{24} \\ f_{33} & f_{34} \end{vmatrix}}{\begin{vmatrix} f_{12} & f_{13} & f_{14} \\ f_{22} & f_{23} & f_{24} \\ 0 & f_{33} & f_{34} \end{vmatrix}},$$

and of course the process is repeated to convergence. This method has efficiency .2934, which is close to the record high value for methods of this class.

3. Methods using derivatives

3.1. Methods using Hermite interpolation

Hindmarsh [38] considers a class of Hermite interpolatory functions. He finds a function $P_n(x)$ satisfying

$$P_n^{(k)}(x_{n-i}) = f^{(k)}(x_{n-i}), \quad (k = 0, \ldots, m_i - 1; i = 1, \ldots, \ell), \tag{3.1}$$

where the m_i are fixed integers, ℓ is the number of previously defined x_{n-i} used in calculating x_n, and the x_{n-i} are previously found estimates for the root ζ. Then x_n is found as a root of $P_n(x) = 0$.

More practically, we define the inverse function

$$F(y) = f^{-1}(y),$$

and fit $Q_n(y)$ to $F(y_i)$ as in (3.1); then we take

$$x_n = Q_n(0).$$

Actually, Hindmarsh finds that the most efficient method of this class has $m_i = 1$ for all i (i.e., it uses no derivatives), and large ℓ. As ℓ converges to ∞, the efficiency converges to $\log 2$.

Kung and Traub [29] propose a family of inverse Hermite interpolatory formulas starting with

$$w_1 = x,$$
$$w_2 = x - \frac{f(x)}{f'(x)},$$
$$w_3 = w_2 - \frac{f(x)f(w_2)}{[f(x) - f(w_2)]^2}\frac{f(x)}{f'(x)};$$

the authors supply an Algol program to construct higher-order methods. w_n has efficiency $\log(2^{1-\frac{1}{n}})$; e.g., $n = 4$ gives .226.

3.2. Multi-point methods

Neta [39] proposes a method having order 10.81 and efficiency .2585. For a fixed triple (w_0, z_0, x_0), he recursively computes triples (w_i, z_i, x_i), $i = 1, 2, \ldots$ as follows:

$$w_i = x_{i-1} - \frac{f(x_{i-1})}{f'(x_{i-1})} + [f(w_{i-1})\phi_z - f(z_{i-1})\phi_w]\frac{f(x_{i-1})^2}{f(w_{i-1}) - f(z_{i-1})},$$

where

$$\phi_w = \frac{w_{i-1} - x_{i-1}}{[f(w_{i-1}) - f(x_{i-1})]^2} - \frac{1}{[f(w_{i-1}) - f(x_{i-1})]f'(x_{i-1})},$$

and ϕ_z is the same, but with z_{i-1} in place of w_{i-1}. Then he obtains

$$z_i = x_{i-1} - \frac{f(x_{i-1})}{f'(x_{i-1})} + [f(w_i)\phi_z - f(z_{i-1})\psi_w]\frac{f(x_{i-1})^2}{f(w_i) - f(z_{i-1})},$$

where

$$\psi_w = \frac{w_i - x_{i-1}}{[f(w_i) - f(x_{i-1})]^2} - \frac{1}{[f(w_i) - f(x_{i-1})]f'(x_{i-1})}.$$

Finally, he computes

$$x_i = x_{i-1} - \frac{f(x_{i-1})}{f'(x_{i-1})} + [f(w_i)\psi_z - f(z_i)\psi_w]\frac{f(x_{i-1})^2}{f(w_i) - f(z_i)},$$

where

$$\psi_z = \frac{z_i - x_{i-1}}{[f(z_i) - f(x_{i-1})]^2} - \frac{1}{[f(z_i) - f(x_{i-1})]f'(x_{i-1})}. \tag{3.2}$$

Bi et al. [40] present a family of eighth-order methods with four evaluations, and hence efficiency $\frac{\log 8}{4} = .2258$ (this is optimum for four evaluations and no memory). A useful member of this family is as follows:

$$y_i = x_i - \frac{f(x_i)}{f'(x_i)},$$
$$z_i = y_i - \frac{2f(x_i) - f(y_i)}{2f(x_i) - 5f(y_i)}\frac{f(y_i)}{f'(x_i)},$$
$$x_{i+1} = z_i - \frac{f(x_i) + (\gamma + 2)f(z_i)}{f(x_i) + \gamma f(z_i)}\frac{f(z_i)}{f[z_i, y_i] + \frac{2(f[z_i,x_i] - f'(x_i))}{z_i - x_i}(z_i - y_i)},$$

where γ is a parameter. The value $\gamma = 1$ gave very good results.

There are several other methods of order 8 with four evaluations in the literature, but we do not mention them here to save space; and it is unlikely that any of them will be substantially better than the one reported above.

Neta [41] gives a method of order 16 with five evaluations, as follows. Let

$$w_i = x_i - \frac{f(x_i)}{f'(x_i)},$$
$$z_i = w_i - \frac{f(w_i)}{f'(x_i)}\frac{f(x_i) + 2f(w_i)}{f(x_i)}.$$

Now let

$$F_\delta = f(\delta_i) - f(x_i),$$
$$\phi_\delta = \frac{\delta_i - x_i}{F_\delta^2} - \frac{1}{F_\delta f'(x_i)},$$

where $\delta = w$ or $\delta = z$ (e.g., if $\delta = w$, $\delta_i = w_i$, $F_\delta = f(w_i) - f(x_i)$).

Next, compute

$$D = \frac{\phi_w - \phi_z}{F_w - F_z}, \quad \gamma = \phi_w - DF_w,$$
$$t_i = x_i - \frac{f(x_i)}{f'(x_i)} + \gamma f^2(x_i) - Df^3(x_i),$$
$$F_t = f(t_i) - f(x_i), \quad \phi_t = \frac{t_i - x_i}{F_t^2} - \frac{1}{F_t f'(x_i)},$$
$$e = \frac{\frac{\phi_t - \phi_z}{F_t - F_z} - \frac{\phi_w - \phi_z}{F_w - F_z}}{F_t - F_w},$$
$$d = \frac{\phi_t - \phi_z}{F_t - F_z} - e(F_t + F_z), \quad c = \phi_t - dF_t - eF_t^2,$$

and finally

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} + cf^2(x_i) - df^3(x_i) + ef^4(x_i).$$

The efficiency is $\log(\sqrt[5]{16}) = .2408$.

Geum and Kim [42] give another family of methods of order 16 with five evaluations (and thus efficiency .2408). They use

$$\begin{aligned}
y_i &= x_i - \frac{f(x_i)}{f'(x_i)},\\
z_i &= y_i - K_f(u_i)\frac{f(y_i)}{f'(x_i)},\\
s_i &= z_i - H_f(u_i, v_i, w_i)\frac{f(z_i)}{f'(x_i)},\\
x_{i+1} &= s_i - W_f(u_i, v_i, w_i, t_i)\frac{f(s_i)}{f'(x_i)},
\end{aligned}$$

where

$$\begin{aligned}
K_f(u_i) &= \frac{1 + \beta u_i + \left(-9 + \frac{5\beta}{2}\right)u_i^2}{1 + (\beta - 2)u_i + (-4 + \beta^2)u_i^2},\\
H_f(u_i, v_i, w_i) &= \frac{1 + 2u_i + (2 + \sigma)w_i}{1 - v_i + \sigma w_i},\\
W_f(u_i, v_i, w_i, t_i) &= \frac{1 + 2u_i + (2 + \sigma)v_i w_i}{1 - v_i - 2w_i - t_i + 2(1 + \sigma)v_i w_i} + G(u_i, w_i).
\end{aligned}$$

Here, β and σ are free parameters, and $G(u, w)$ is an analytic function, while

$$u_i = \frac{f(y_i)}{f(x_i)}, \qquad v_i = \frac{f(z_i)}{f(y_i)}, \qquad w_i = \frac{f(z_i)}{f(x_i)}, \qquad t_i = \frac{f(s_i)}{f(z_i)}.$$

The authors experimented with several values of (β, σ), and had most success with $\beta = 2, \sigma = -2$, and

$$G(u, w) = -\frac{1}{2}[uw[6 + 12u + u^2(24 - 11\beta) + u^3\phi_1 + 4\sigma]] + \phi_2 w^2,$$

where

$$\phi_1 = 11\beta^2 - 66\beta + 136; \qquad \phi_2 = 2u(\sigma^2 - 2\sigma - 9) - 4\sigma - 6.$$

Alefeld and Potra [43] present another bracketing method as follows. Assume that the interval $[a, b]$ contains a root ζ. Let $y_0 = a$ and $z_0 = b$, and for $i = 0, 1, 2, \ldots$ compute

$$\begin{aligned}
y_{i+1} &= y_i - \Delta f(y_i, z_i)^{-1} f(y_i),\\
\overline{z}_{i+1} &= y_i - \frac{f(y_i)}{\Delta f(y_i, y_{i+1})},\\
z_{i+1} &= \min\{\overline{z}_{i+1}, z_i\},
\end{aligned}$$

where $\Delta f(s, t)$ is the divided difference of $f(x)$ at the points s and t. It is proved that y_i and $z_i \to \zeta$ from below and above. The efficiency is only .191, but again convergence is guaranteed.

3.3. Methods using quadratics

King [44] describes the "Tangent parabola" method as follows. Let

$$\begin{aligned}
a_0 &= f_0 - f_2 + f_1'(x_2 - x_0),\\
b_0 &= 2x_1(f_2 - f_0) + f_1'(x_0^2 - x_2^2),\\
c_0 &= x_2(2x_1 - x_2)f_0 + x_0(x_0 - 2x_1)f_2 + x_0x_2(x_2 - x_0)f_1',
\end{aligned}$$

where

$$f_i = f(x_i) \quad \text{and} \quad f_i' = f'(x_i)\ (i = 0, 1, 2, \ldots).$$

Then we compute

$$x_3 = \frac{1}{2}\left(x_2 + \frac{-b_0 \pm \sqrt{b_0^2 - 4a_0c_0}}{2a_0}\right).$$

Also, let

$$A_1 = f_1' - f_3'; \qquad B_1 = 2(x_1f_3' - x_3f_1'),$$
$$C_1 = 2(x_1 - x_3)f_2 - x_2^2A_1 - x_2B_1,$$

and then

$$x_4 = \frac{-B_1 \pm \sqrt{B_1^2 - 4A_1C_1}}{2A_1}.$$

The process may of course be repeated until convergence.

The convergence order is 3, and two evaluations are required per iteration, so the efficiency is .238.

Costabile et al. [45] give a method for real roots based on quadratic interpolation, with successive approximations bracketing the root (as in bisection or Regula Falsi). The iteration used is

$$x_{i+1} = x_i - \frac{2f_i}{f_i' \pm \sqrt{f_i^2 - 4\sigma_i f_i}}, \tag{3.3}$$

where

$$\sigma_i = \frac{f_{i-1} - f_i - f_i'(x_{i-1} - x_i)}{(x_{i-1} - x_i)^2}.$$

Select x_0 and x_1 so that

$$f(x_0)f(x_1) < 0, \tag{3.4}$$

and choose the sign in (3.3) so that x_2 is in $[x_0, x_1]$ (it is proved that this exists and is unique). Then define the new x_1 as x_0 or the old x_1 so that (3.4) remains true. Finally, we iterate to convergence, which is *guaranteed*. The efficiency is .190, lower than that of most methods considered in this work; but it is mentioned because of its valuable feature of guaranteed convergence.

The later paper Costabile et al. [46] presents a method similar to Muller's except that they fit a quadratic through the two latest approximations and the point mid-way between them. The authors also use bracketing as before to ensure global convergence to a real root. The order is 2 and the efficiency .3105, among the highest known for this type of method.

4. Companion matrix methods

4.1. Algorithms based on repeated squaring

Hereafter, we write

$$p(x) = x^n + c_{n-1}x^{n-1} + \cdots + c_1x + c_0. \tag{4.1}$$

We define the companion matrix of this polynomial as

$$\mathbf{C} = \begin{bmatrix} 0 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & \cdots & \cdots & 0 & 0 & 1 \\ -c_0 & -c_1 & \cdots & \cdots & -c_{n-2} & -c_{n-1} \end{bmatrix}, \tag{4.2}$$

and then

$$\det(x\mathbf{I} - \mathbf{C}) = p(x) \tag{4.3}$$

(see [47]), and so we can find polynomial roots as the eigenvalues of this special matrix. This leads to companion matrix methods; they can be extended to the case of nonmonic polynomials (see [48,49]).

Stewart [50] and Householder [51] extend the work of Sebastiao e Silva [52] by starting with an arbitrary nontrivial polynomial $h_0(x)$ of degree less than n, and applying

$$h_{i+1} = [h_i]^2 \bmod p(x), \quad i = 0, 1, \ldots.$$

Then they employ companion matrix methods.

In particular, Stewart shows that, if

$$|h_0(r_1)| > |h_0(r_i)| \quad (r_i \neq r_1),$$

then $\frac{h_i(x)}{h_i(0)}$ converges to $\frac{\pi_1(x)}{\pi_1(0)}$, where

$$\pi_1(x) = \frac{p(x)}{x - r_1}, \quad \text{so } x - r_1 = \frac{p(x)}{\pi_1(x)}.$$

Finally, he shows that

$$\mathbf{h}_i = [h_0(\mathbf{C})]^{2^i} \begin{bmatrix} 1 \\ 0 \\ \cdot\cdot \\ \cdot\cdot \\ 0 \end{bmatrix},$$

where by $\mathbf{h}_i$ he denotes the coefficient vector of $h_i(x)$. For a simple root, the convergence is quadratic, but the efficiency is low. The transition from h_i to h_{i+1} requires order $n \log n$ flops, that is, order $\log n$ function evaluations. This feature persists in the amended variants of this approach in [53,54].

4.2. Companion and generalized companion matrix eigen-solvers based on QR and LR algorithms

Another group of the companion matrix methods incorporates the powerful QR eigenvalue algorithm and its variations. The local efficiency of these algorithms is not at the highest level, but we include them because they have good empirical global convergence (right from the start), even though the known formal estimates do not match such an excellent empirical property.

Matlab incorporates the classical QR algorithm, which takes order n^2 flops per iteration. Uhlig [55] proposes its practical improvement for polynomial root-finding, which he calls the DQR algorithm. Such a variation on the QR method (see [56]) still takes order n^2 flops per iteration; this corresponds to order n function evaluations per iteration. He applies the Euclidean Algorithm to find a tridiagonal matrix whose characteristic polynomial is $p(z)$. He then transforms the given unsymmetric tridiagonal matrix to a complex symmetric one, and applies Givens transformations to implement the QR method (for more details see Uhlig's papers or [7, Chapter 6]). Uhlig has written a program *pzero* based on his method; it is available from his web-site at www.auburn.edu/~uhligfd/m_files/pzero.m.

Bini et al. [57,58] stay with the QR algorithm but exploit the rank structure of the companion and diagonal + rank-one (hereafter DPR1) generalized companion input matrices to decrease the number of flops per iteration step to cn for a scalar c provided all the eigenvalues are real. Eigen-solving for DPR1 matrices is equivalent to root-finding for the important secular equations (see the details in [57], and the references therein).

To define the latter matrices, assume n distinct values $s_1, \ldots, s_n$; then define a rank-one matrix $\mathbf{E}_d$ with diagonal entries

$$d_i = \frac{p(s_i)}{q_i(s_i)},$$

where

$$q_i(x) = \prod_{j \neq i} (x - s_j),$$

and finally define an associated generalized companion matrix

$$\mathbf{C} = \mathbf{D}_s - \mathbf{E}_d, \tag{4.4}$$

where $\mathbf{D}_s = \begin{bmatrix} s_1 & 0 & \cdot\cdot \\ \cdot\cdot & \cdot\cdot & \cdot\cdot \\ \cdot\cdot & 0 & s_n \end{bmatrix}$ (i.e., it is a diagonal matrix with the (i, i) element s_i) and $\mathbf{E}_d$ of rank one is arbitrary; Elsner [59] proposes

$$\mathbf{E}_d = \begin{bmatrix} d_1 & d_2 & \cdot\cdot & d_n \\ d_1 & d_2 & \cdot\cdot & d_n \\ \cdot\cdot & \cdot\cdot & \cdot\cdot & \cdot\cdot \\ d_1 & d_2 & \cdot\cdot & d_n \end{bmatrix}.$$

Then (4.3) still holds for $\mathbf{C}$ of (4.4), that is, $\mathbf{C}$ is indeed a generalized companion matrix. This can be readily deduced based on Lagrange interpolation (see [59] or [60]).

The restriction required in [57,58] that the input matrix should have only real eigenvalues has been removed in a number of subsequent algorithms (see [61–65]). The paper [66] proposes further practical acceleration based on replacing the QR by the LR scheme for a restricted class of totally positive input matrices. The number of flops per iteration is still of order cn, where c varies depending on the selected representation for structured matrices. It has order n under the known numerically stable representations; by relaxing this requirement, one can decrease it to a constant, which, however, is still too large to make the resulting algorithms competitive for root-refining.

4.3. Companion and generalized companion matrix eigen-solvers based on the inverse power iteration and its variations

Bini, Gemignani and Pan in the paper [67] apply the inverse power method to companion and DPR1 generalized companion matrices.

The inverse power method for a general matrix $\mathbf{C}$ is defined as follows. Let $z^{(0)}$ be a sufficiently close approximation to an eigenvalue z_j of $\mathbf{C}$, and let

$$\mathbf{v} = \sum_{k=1}^{n} a_k \mathbf{v}_k$$

with $\|\mathbf{v}\|_2 = 1$, where $\mathbf{v}_k$, $(k = 1, \ldots, n)$ are the eigenvectors of $\mathbf{C}$ and $a_k \neq 0$. Let

$$\begin{aligned}
&\mathbf{x}^{(0)} = \mathbf{v},\\
&\mathbf{y}^{(i)} = (\mathbf{C} - z^{(i-1)}\mathbf{I})^{-1}\mathbf{x}^{(i-1)},\\
&\mathbf{x}^{(i)} = \frac{\mathbf{y}^{(i)}}{\|\mathbf{y}^{(i)}\|_2},\\
&z^{(i)} = \mathbf{x}^{(i)T}\mathbf{C}\mathbf{x}^{(i)}.
\end{aligned}$$

Solution of all the above equations is repeated for $i = 1, 2, \ldots$.

Then the pairs $(\mathbf{y}^{(i)}, z^{(i)})$ quadratically converge to an eigenvector/eigenvalue pair $(\mathbf{v}_j, z_j)$ (under certain conditions). Section 3 of the paper [67] describes several methods of choosing the initial z. It is estimated that for $\mathbf{C}$ in (4.4) the vector product $(\mathbf{C} - z\mathbf{I})^{-1}\mathbf{x}$ can be computed in $9n$ flops, whereas $7n$ flops suffice in the case of $\mathbf{C}$ in (4.2); clearly in both cases the vector $C\mathbf{x}$ can be computed even faster. This method is most attractive where only a few roots of $p(x)$ are required. A program IPSOLVE implementing it can be found at the web-site www.bezout.dm.unipi.it.

Hereafter, we assume that a function evaluation takes $2n \pm O(1)$ flops.

Pan and Zheng [68] propose and test two nontrivial extensions where the flops bounds per iteration are decreased to $4n + 1$ and $3n + 4$ in the case of companion matrices $\mathbf{C}$ in (4.2) (this can be interpreted as 2 and 1.5 function evaluations, which implies the efficiencies $\frac{1}{2}\log 2 = .1505$ and $\frac{2}{3}\log 2 = .2007$, respectively), and to $5n + 1$ and $4n + 2$ flops in the case of DPR1 generalized companion matrices $\mathbf{C}$ in (4.4) (this can be interpreted as 2.5 and 2 function evaluations, which implies the efficiencies $\frac{2}{5}\log 2 = .1204$ and $\frac{1}{2}\log 2 = .1505$, respectively).

5. Simultaneous methods

5.1. WDK and Aberth algorithms

The heading of this section refers to numerous methods that combine n approximations to all the n roots to refine them simultaneously. Milovanovic and Petkovic [69] compare ten such methods for polynomials of moderate degrees n where no special initial information apart from the coefficients is assumed, and conclude that the best of this class is the Gauss–Seidel version of the Weierstrass–Durand–Kerner (hereafter WDK) method (see [70–72]). Let $z_i^{(k)}$ be denoted by z_i and $z_i^{(k+1)}$ by $\hat{z}_i$ (where k is the iteration number and i is the index of the root towards which z_i is supposedly converging). Then the method in question uses

$$\hat{z}_i = z_i - \frac{p(z_i)}{\prod_{j=1}^{i-1}(z_i - \hat{z}_j)\prod_{j=i+1}^{n}(z_i - z_j)} \quad (i = 1, \ldots, n). \tag{5.1}$$

For large n, however, Petkovic and Milovanovic [73] find the Gauss–Seidel Improved Ehrlich–Aberth method best, namely

$$\hat{z}_i = z_i - \frac{1}{\frac{p'(z_i)}{p(z_i)} - \sum_{j=1}^{i-1}\frac{1}{z_i - \hat{z}_j} - \sum_{j=i+1}^{n}\frac{1}{z_i - z_j + \frac{p(z_j)}{p'(z_j)}}}. \tag{5.2}$$

Bini and Fiorentino, extending Bini [74], have written a robust program called MPSOLVE (see [75]). It implements Ehrlich–Aberth's method (see [76,77]), namely

$$\hat{z}_i = z_i - \frac{1}{\frac{p'(z_i)}{p(z_i)} - \sum_{j=1}^{i-1}\frac{1}{z_i - \hat{z}_i} - \sum_{j=i+1}^{n}\frac{1}{z_i - z_j}}. \tag{5.3}$$

It uses multiprecision and can give the multiplicity of the roots. It is available at the web-site www.bezout.dm.unipi.it.

It never failed on 1000 polynomials of degree up to 25,000. The program Polzeros is a slightly different implementation of Ehrlich–Aberth's method, also to be found at the above web-site.

Iterations (5.1)–(5.3) approximate all n roots by using $4n^2 \pm O(n)$, $8n^2 \pm O(n)$ and $6n^2 \pm O(n)$ flops per iteration loop, respectively, and converge (locally) with order 2, 3, and 3, respectively, and so the cost of an iteration loop is translated into $2n \pm O(1)$, $4n \pm O(1)$,and $3n \pm O(1)$ function evaluations, which is roughly 2, 4, and 3 per root, respectively. This implies quite low efficiencies $\frac{1}{2}\log 2 = .1505$, $\frac{1}{4}\log 3 = .119$, and $\frac{1}{3}\log 3 = .159$, respectively.

5.2. Simultaneous methods based on refinement of polynomial factorization

Some other simultaneous methods rely on recursive refinement of polynomial factorization, studied in [9,78,10–13, 79,8]. Next, we outline some of these techniques. Further details can be found in the above papers and in Chapter 15 of [5].

Suppose we are given the coefficients of a monic polynomial

$$p(x) = \sum_{i=0}^{n} p_i x^i = p_n(x - x_1)\cdots(x - x_n)$$

for $n \neq 0$ and sufficiently close initial approximations $z_1, \ldots, z_n$ to its n simple roots $x_1, \ldots, x_n$, supplied, for example, by the WDK or Ehrlich–Aberth algorithms.

This defines an approximate factorization

$$p(x) \approx f(x) = p_n(x - z_1)\cdots(x - z_n),$$

and conversely the algorithms of Schönhage [9] extending Ostrowski [80,81] readily recover approximate roots from such a factorization. Now, our task is the refinement of a crude initial factorization. We recursively apply Newton's multivariate iteration, assuming for simplicity that $p(x)$ is monic, $p_n = 1$.

Iteration recursively updates the linear factors

$$f_1 = x - z_1, \ldots, f_n = x - z_n$$

to produce new factors

$$f_1^{(\text{new})} = f_1 + t_1, \ldots, f_n^{(\text{new})} = f_n + t_n.$$

Here, the scalars $t_1, \ldots, t_n$ define Newton's corrections and satisfy the partial fraction decomposition (hereafter PFD)

$$\frac{r}{f} = \frac{t_1}{f_1} + \cdots + \frac{t_n}{f_n},$$
$$r = r(x) = p - f, \quad p = p(x) \text{ and } f = f(x).$$

Consequently,

$$t_i = \frac{r}{q_i} \bmod f_i = \frac{r(z_i)}{q_i(z_i)}$$

for

$$q_i = \frac{f}{f_i}, \quad i = 1, \ldots, n.$$

We can rewrite these expressions as

$$z_i^{(\text{new})} = z_i - \frac{p(z_i)}{q_i(z_i)}, \quad i = 1, \ldots, n,$$

and arrive at the WDK classical iteration (without the Gauss–Seidel updating). Its convergence order is still 2, and its efficiency is still $\frac{1}{2}\log 2 = .1505$. It may be surprising that the simple link between computing the factorization $p(x) \approx f(x) = p_n(x - z_1)\cdots(x - z_n)$ and the WDK algorithm had not been observed until Pan and Zheng [8].

5.3. Simultaneous divide and conquer methods based on recursive polynomial factorization

It may be even more surprising that the efficiency of the factorization approach can dramatically increase where it is recast in the divide and conquer iterative process covered in [10–13] and Chapter 15 of [5]. Indeed, assume a sufficiently close initial approximation to a *balanced* factorization of the polynomial p into the product of two factors of comparable degrees,

$$p \approx q = f_1 f_2 \tag{5.4}$$

where

$$c_- \leq \deg f_1 / \deg f_2 \leq c_+ \tag{5.5}$$

for two positive constants c_- and c_+, and then recursively update it; namely, extend the above construction by writing

$$f_i^{(\text{new})} = f_i + t_i$$

for $i = 1, 2$ and Newton's correction polynomials t_i satisfying

$$\frac{r}{q} = \frac{t_1}{q_1} + \frac{t_2}{q_2}, \tag{5.6}$$

where $\deg t_j < \deg f_j$ and $q_j = \frac{q}{f_j}$ for $j = 1, 2$.

We can compute this PFD by using $O(n \log^2 n)$ flops (see, for example, [82], Problem 4.2c (PART·FRAC), pp. 30–31).

The iterative updating process ends with sufficiently close approximations of the factors f_1 and f_2. Then we can recursively factorize both of them in similar fashion until we arrive at the complete approximate factorization

$$p(x) = \sum_{i=0}^{n} p_i x^i \approx p_n(x - \tilde{x}_1) \cdots (x - \tilde{x}_n),$$

where $\tilde{x}_1 \approx x_1, \ldots, \tilde{x}_n \approx x_n$ approximate the n distinct roots of p within required accuracy. An effective algorithm for computing an initial approximate factorization has been combined with such a refinement process in [10–13] to approximate all n complex roots of a polynomial by using record and nearly optimal number of bitwise (Boolean) operations.

Now, represent the above recursive process of refining the factorization by a binary tree with a root p, which has two children f_1 and f_2; each of them in turn has two children such that $f_1 \approx f_{11} f_{12}$ and $f_2 \approx f_{21} f_{22}$, etc. At every level of the tree, its nodes represent polynomials whose degrees sum to $n - l$, where l denotes the number of linear factors output at the previous levels. The tree has $O(\log n)$ levels due to bound (5.5). It follows that computing Newton's corrections for all factor polynomials at each level takes $O(n \log^2 n)$ flops.

This is translated into $O(\frac{\log^2 n}{n})$ function evaluations and the efficiency $\frac{n \log q}{c' \log^2 n}$, where $c' \neq 0$ is a fixed constant and q is the order of convergence. Newton's iteration has local quadratic convergence, and the substitution of $q = 2$ turns the estimated efficiency into

$$\frac{n}{c'' \log^2 n}$$

for a positive constant $c'' = c'/\log 2$; then the efficiency grows to infinity as the degree n increases to infinity.

The divide and conquer factorization has practical limitations. The factors readily lose sparseness of an input polynomial and may have much larger coefficients. A further modification is needed to avoid numerically unstable computation of the PFDs.

6. Conclusions and further work

The problem of polynomial root-refining that starts with some crude but reasonably close initial approximations to the roots is practically important. It can be effective to specialize root-refiners at this local stage by choosing them independently of the global methods that supply crude initial approximations.

With this motivation, we have surveyed a number of iterative root-refiners and estimated their efficiency, which in our definition is inversely proportional to the arithmetic time of the solution (that is to the number of flops involved) and which indeed naturally measures the efficiency of a root-refining process. According to our measure, which is just a variation of a customary concept of numerical analysis, the most efficient iterative methods for refining a single root appear to be those by Aitken [36], Hindmarsh [38], Jarratt [23], Kogan et al. [27], Larkin [32,33], and Costabile et al. [46], although their lead over some other methods is rather narrow. It is most surprising that the simultaneous refinement of all roots based on divide and conquer polynomial factorization can dramatically increase the efficiency; namely, we can make the efficiency grow to infinity as the degree n of the polynomial grows to infinity. This observation should be investigated further, both formally and experimentally, possibly with the incorporation of the techniques from Chapter 15 of [5] and [8].

Table 1 summarizes our estimates of Sections 2 and 3 for the efficiency of refining a single root. All the estimates are bounded by .3010 and for large degree n are dramatically less than the efficiency $\frac{n}{c'' \log^2 n}$ for a positive constant c'' (obtained at the end of Section 5). Other estimates of Sections 4 and 5 are less than .2010 and are too low to compete even with cited record estimate .3010. We cover the supporting algorithms in Sections 4 and 5 because they are technically important and have very good empirical global convergence, but we do not include the respective estimates in the table.

Acknowledgement

The second author was supported by NSF Grant CCF-1116736 and PSC CUNY Awards 63153–0041 and 64512–0042.

Table 1
Efficiency of polynomial root-refining.

AUTHOR(S)	DATE	SOURCE	EFFY	REAL ONLY
Muller	1956	Math. Tab. Aids Comput. 10 208	.265	
Sharma	2004	Comput. Math. Appl. 48 709	.265	
Jarratt	1970	Num. Meths. Nonlin. Alg. Equs. 1–26	.2975	
Anderson/Björck	1973	BIT 13 253	.233	yes
King	1976	Computing 17 49	.265	yes
Kung/Traub	1974	J. Ass. Comp. Mach. 21 643	→.3010	
Jarratt/Nudds	1965	Comput. J. 8 62	.265	yes
Kogan	1966	Tashkent Gos. Nauen Trudy Vyp. 276 53	.265	
Kogan et al.	2007	Appl. Math. Comp. 188 75	→.3010	
Larkin	1980	Math. Comp. 35 803	→.3010	
Larkin	1981	Numer. Math. 37 93	→.3010	
Aitken	1926	Proc. Roy. Soc. Edin. 46 289	.3010	
Kalantari	2000	J. Comp. Appl. Math. 126 287	.2934	
Hindmarsh	1972	SIAM J. Num. Anal. 9 205	→.3010	
Neta	1983	J. Comput. Math. 14 191	.2585	
King	1973	Numer. Math. 18 298	.238	
Bi et al.	2009	Appl. Math. Comput. 214 236	.2258	
Neta	1981	J. Comput. Math. 9 353	.241	
Geum/Kim	2011	J. Comput. Appl. Math. 235 3178	.2408	
Costabile et al.	2001	Numer. Algs. 28 87	.190	yes
Costabile et al.	2006	Calcolo 43 39	.3010	
Alefeld/Potra	1988	ZAMM 68 331	.190	yes

References

[1] E.T. Bell, The Development of Mathematics, McGraw-Hill, New York, 1940.
[2] C.A. Boyer, A History of Mathematics, Wiley, New York, 1968.
[3] V.Y. Pan, Solving a polynomial equation: some history and recent progress, SIAM Review 39 (2) (1997) 187–220.
[4] V.Y. Pan, Solving polynomials with computers, American Scientist 86 (1998) 1998.
[5] J.M. McNamee, V.Y. Pan, 2012, Numerical Methods for Roots of Polynomials, Part 2, 780 + XIX pages, submitted for publication by Elsevier publishers.
[6] J.M. McNamee, A 2002 update of the supplementary bibliography on root of polynomials, J. Comput. Appl. Math. 142 (2002) 433–434. also at web-site www.yorku.ca/~mcnamee/.
[7] J.M. McNamee, Numerical Methods for Roots of Polynomials (Part 1), Elsevier, Amsterdam, 2007.
[8] V.Y. Pan, A.-L. Zheng, 2011, Root-finding by expansion with independent constraints, Computers and Mathematics (with Applications) 62, 62, pp. 3164–3182, Proceedings version is in Proceedins of International Symposium on Symbolic-Numerical Computations (SNC 2011), San Jose, California, June 2011, edited by Marc Moreno Masa, ACM Press, New York.
[9] A. Schönhage, The Fundamental Theorem of Algebra in Terms of Computational Complexity, Department of Math., University of Tübingen, Tübingen, Germany, 1982.
[10] V.Y. Pan, 1995, Optimal up to polylog factors sequential and parallel algorithms for approximating complex polynomial zeros, in: Proc. 27th Ann. ACM Symp. on Theory of Computing, pp. 741–750, ACM Press, New York.
[11] V.Y. Pan, Optimal and nearly optimal algorithms for approximating polynomial zeros, Computers and Math. (with Applications) 31 (12) (1996) 97–138.
[12] V.Y. Pan, 2001, Univariate polynomials: nearly optimal algorithms for factorization and rootfinding, in: Proc. International Symposium on Symbolic and Algebraic Computation (ISSAC '01), pp. 253–267, ACM Press, New York.
[13] V.Y. Pan, Univariate polynomials: nearly optimal algorithms for factorization and rootfinding, J. Symbolic Computations 33 (5) (2002) 701–733.
[14] J. Renegar, On the worst-case arithmetic complexity of approximating zeros of polynomials, J. Complexity 3 (1987) 90–113.
[15] M.-H. Kim, 1985, Computational complexity of the Euler type algorithms for the roots of complex polynomials, PhD Thesis, City University of New York.
[16] S. Smale, Newton's method estimates from data at one point, in: R.E. Ewing, K.I. Cross, C.F. Martin (Eds.), The Merging Disciplines: New Directions in Pure, Applied and Computational Math., Springer, 1986, pp. 185–196.
[17] J.H. Curry, On zero finding methods of higher order from data at one point, J. Complexity 5 (1989) 219–237.
[18] M.S. Petkovic, D. Herceg, Point estimation of simultaneous methods for solving polynomial equations: a survey, J. Comput. Appl. Math. 136 (2001) 183–207.
[19] D.E. Muller, A method for solving algebraic equations using an automatic computer, Math. Tables Aids Comput. 10 (1956) 208–215.
[20] A.S. Householder, The Numerical Treatment of a Single Nonlinear Equation, McGraw-Hill, New York, 1970.
[21] I. Barrodale, K.B. Wilson, A Fortran program for solving a non-linear equation by Muller's method, J. Comput. Appl. Math. 4 (1978) 159–166.
[22] J.R. Sharma, A family of methods for solving nonlinear equations using quadratic interpolation, Comput. Math. Appls. 48 (2004) 709–714.
[23] P. Jarratt, Nonlinear equations in one variable, in: P. Rabinowitz (Ed.), Numerical Methods for Nonlinear Algebraic Equations, Gordon and Breach, London, 1970, pp. 1–26.
[24] N. Andersen, A. Björck, A new high order method of Regula Falsi type for computing a root of an equation, BIT 13 (1973) 253–264.
[25] R.F. King, Methods without secant steps for finding a bracketed root, Computing 17 (1976) 49–57.
[26] T.I. Kogan, Generalization of the method of chords for an algebraic or transcendental equation, Tashkent Gos. Univ. Nauen. Trudy Vyp. 276 (1966) 53–56. (in Russian).
[27] T. Kogan, L. Sapir, A. Sapir, A nonstationary iterative second-order method for solving nonlinear equations, Appl. Math. Comput. 188 (2007) 75–82.
[28] J.F. Traub, Optimal iterative processes: theorems and conjectures, Proc. 1971 IFIP Congress (1971) 1273–1277. Booklet TA-1.
[29] H.T. Kung, J.F. Traub, Optimal order of one-point and multipoint iterations, J. Assoc. Comput. Mach. 21 (1974) 643–651.
[30] A.G. Werschultz, Maximal order for multipoint methods with memory using Hermitian information, Intern. J. Comput. Math. Sec. B 9 (1981) 223–241.
[31] P. Jarratt, D. Nudds, The use of rational functions in the iterative solution of equations on a digital computer, Computer J. 8 (1965) 62–65.
[32] F.H. Larkin, Root-finding by fitting rational functions, Math. Comp. 35 (1980) 803–816.
[33] F.H. Larkin, Root finding by divided differences, Numer. Math. 37 (1981) 93–104.
[34] V. Norton, Algorithm 631: finding a bracketed zero by Larkin's method of rational interpolation, ACM Trans. Math. Softw. 11 (1985) 120–134.
[35] P. Henrici, Elements of Numerical Analysis, Wiley, New York, 1964.
[36] A.C. Aitken, On Bernoulli's numerical solution of algebraic equations, Proc. Roy. Soc. Edin. 46 (1926) 289–305.

[37] B. Kalantari, Generalization of Taylor's theorem and Newton's method via a new family of determinantal interpolation formulas and its applications, J. Comput. Appl. Math. 126 (2000) 287–318.
[38] A.C. Hindmarsh, Optimality in a class of root-finding algorithms, SIAM J. Numer. Anal. 9 (1972) 205–214.
[39] B. Neta, A new family of higher order methods for solving equations, Int. J. Comput. Math. 14 (1983) 191–196.
[40] W. Bi, Q. Wu, H. Ren, A new family of eighth-order iterative methods for solving non-linear equations, Appl. Math. Comput. 214 (2009) 236–245.
[41] B. Neta, On a family of multipoint methods for non-linear equations, Int. J. Comput. Math. 9 (1981) 353–361.
[42] Y.H. Geum, Y.I. Kim, A biparametric family of optimally convergent sixteenth-order multipoint methods with their fourth-step weighting function as a sum of a rational and a generic two-variable function, J. Comput. Appl. Math. 235 (2011) 3178–3188.
[43] G. Alefeld, F.A. Potra, On two higher-order enclosing methods of J.W. Schmidt, Zeit. Ang. Math. Mech. 68 (1988) 331–337.
[44] R.F. King, Tangent methods for nonlinear equations, Numer. Math. 18 (1972) 298–304.
[45] F. Costabile, M.I. Gualtieri, R. Luceri, A new iterative method for the computation of the solutions of nonlinear equations, Numer. Algs. 28 (2001) 87–100.
[46] F. Costabile, M.I. Gualtieri, R. Luceri, A modification of Mullers method, Calcolo 43 (1) (2006) 39–50.
[47] L. Brand, The companion matrix and its properties, Amer. Math. Monthly 71 (1964) 629–634.
[48] C.F. Jónsson, S. Vavasis, Solving polynomials with small leading coefficients, SIAM J. Matrix Anal. Appl. 26 (2) (2005) 400–412.
[49] R.M. Corless, On a generalized companion matrix pencil for matrix polynomials expressed in the Lagrange basis, in: Dongming Wang, Lihong Zhi (Eds.), Symbolic-Numeric Computation, Birkhäuser, Basel/Boston, 2007, pp. 1–15.
[50] G.W. Stewart, Short notes: on the convergence of Sebastiao E Silva's method for finding a zero of a polynomial, SIAM Rev. 12 (1970) 458–460.
[51] A.S. Householder, Generalization of an algorithm by Sebastiao e Silva, Numerische Math. 16 (1971) 375–382.
[52] J. Sebastiao e Silva, Sur une méthode d'Approximation semblable a celle de graeffe, Portugal Math. 2 (1941) 271–279.
[53] J.P. Cardinal, On two iterative methods for approximating the roots of a polynomial, in: J. Renegar, M. Shub, S. Smale (Eds.), Lectures in Applied Mathematics, in: Proceedings of AMS-SIAM Summer Seminar: Mathematics of Numerical Analysis: Real Number Algorithms, vol. 32, American Mathematical Society, Providence, Rhode Island, 1995, pp. 165–188. Park City, Utah.
[54] V.Y. Pan, Amended DSeSC power method for polynomial root-finding, Computers and Math. (with Applications) 49 (9–10) (2005) 1515–1524.
[55] F. Uhlig, General polynomial roots and their multiplicities in $O(n)$ memory and $O(n^2)$ time, Lin. Mult. Alg. 46 (1999) 327–359.
[56] F. Uhlig, The DQR algorithm, basic theory, convergence and conditional stability, Numer. Math. 76 (1997) 515–553.
[57] D.A. Bini, L. Gemignani, V.Y. Pan, Fast and stable QR eigenvalue algorithms for generalized companion matrices and secular equation, Numerische Math., 3 (2005) 373–408. Also Technical Report 1470, Department of Math., University of Pisa, Pisa, Italy, July 2003.
[58] D.A. Bini, L. Gemignani, V.Y. Pan, Improved initialization of the accelerated and robust QR-like polynomial root-finding, Electron. Trans. Numer. Anal. 17 (2004) 195–205. Proc. version in CASC'2004.
[59] L. Elsner, A remark on simultaneous inclusions of the zeros of a polynomial by Gershgorin's theorem, Numer. Math. 21 (1973) 426–427.
[60] C. Carstensen, Linear construction of companion matrices, Lin. Alg. Appl. 149 (1991) 191–214.
[61] D.A. Bini, F. Daddy, L. Gemignani, On the shifted QR iteration applied to companion matrices, Electron. Trans. Numer. Anal. 18 (2004) 137–152.
[62] D.A. Bini, Y. Eidelman, L. Gemignani, I. Gohberg, A fast QR eigenvalue algorithm for Hessenberg matrices which are rank-one perturbations of unitary matrices, SIAM J. Matrix Anal. Appl. 29 (2) (2007) 566–585.
[63] D.A. Bini, P. Boito, Y. Eidelman, L. Gemignani, I. Gohberg, A fast implicit QR eigenvalue algorithm for companion matrices, Linear Algebra Appl. 432 (2010) 2006–2031.
[64] S. Chandrasekaran, M. Gu, J. Xia, J. Zhu, A fast QR algorithm for companion matrices, Operator Theory Adv. Appl. 179 (2007) 111–143.
[65] M. Van Barel, R. Vandebril, P. Van Dooren, K. Frederix, Implicit double shift QR-algorithm for companion matrices, Numer. Math. 116 (2010) 177–212.
[66] R. Bevilacqua, E. Bozzo, G.M. Del Corso, *qd*-type methods for quasiseparable matrices, SIAM J. Matrix Anal. Appl. 3 (32) (2011) 722–747.
[67] D.A. Bini, L. Gemignani, V.Y. Pan, Inverse Power and Durand–Kerner iterations for univariate polynomial root-finding, Comput. Math. Appl. 47 (2004)447–459. Also Technical Report TR 2002 020, CUNY Ph.D., Program in Computer Science, Graduate Center, City University of New York, 2002.
[68] V.Y. Pan, A.-L. Zheng, New progress in real and complex polynomial root-finding, Computers and Math. (with Applications) 61 (2011) 1305–1334.
[69] G.V. Milovanovic, M.S. Petkovic, On computational efficiency of the iterative methods for the simultaneous approximation of polynomial zeros, ACM Trans. Math. Softw. 12 (1986) 295–306.
[70] K. Weierstrass, Neuer Beweis des fundamentalsatzes der algebra, in: Mathematische Werker, Mayer und Müller, Berlin, 1903, pp. 251–269. Tome III.
[71] E. Durand, Solutions Numeriques des Equations Algebriques, in: Equations du type $F(x) = 0$: racines d'un polynome, 1, Masson, Paris, 1960.
[72] I.O. Kerner, Ein gesamtschrittverfahren zur berechnung der nullstellen von polynomen, Numer. Math. 8 (1966) 290–294.
[73] M.S. Petkovic, G.V. Milovanovic, Computational efficiency of the simultaneous methods for finding polynomial zeros: comparison of various algorithms, in: D. Herceg (Ed.), Numerical Methods and Approximation Theory, University of Novi Sad, 1985, pp. 89–93.
[74] D.A. Bini, Numerical computation of polynomial zeros by means of Aberth's method, Numer. Algs. 13 (1996) 179–200.
[75] D.A. Bini, G. Fiorentino, Design, analysis and implementation of a multiprecision polynomial rootfinder, Numer. Algs. 23 (2000) 127–173.
[76] L.W. Ehrlich, A modified Newton method for polynomials, Comm. ACM 10 (1967) 107–108.
[77] O. Aberth, Iteration methods for finding all zeros of a polynomial simultaneously, Math. Comp. 27 (122) (1973) 339–344.
[78] C.A. Neff, J.H. Reif, 1994, An $o(n^{1+\epsilon})$ algorithm for the complex root problem, Proc. 35th Ann. IEEE Symp. on Foundations of Computer Science, pp. 540–547, IEEE Computer Society Press.
[79] P. Kirrinnis, Polynomial factorization and partial fraction decomposition by simultaneous Newton's iteration, J. Complexity 14 (1998) 378–444.
[80] A.M. Ostrowski, Recherches sur la méthode de Graeffe et les zéros des polynomes et des series de Laurent, Acta Math. 72 (1940) 99–257.
[81] A.M. Ostrowski, Solution of Equations and Systems of Equations, second ed., Academic Press, New York, 1966.
[82] D. Bini, V.Y. Pan, Polynomial and Matrix Computations, Vol. 1: Fundamental Algorithms, Birkhäuser, Boston, 1994.