A Computational Comparison of the First Nine Members of a Determinantal Family of Rootfinding Methods

Bahman Kalantari and Seungyoung Park
Department of Computer Science
Rutgers University, New Brunswick, NJ 08903

Abstract

For each natural number m greater than one, and each natural number k less than or equal to m, there exists a rootfinding iteration function, $B_m^{(k)}$ defined as the ratio of two determinants that depend on the first $m-k$ derivatives of the given function. This infinite family is derived in [4] and its order of convergence is analyzed in [5]. In this paper we give a computational study of the first nine rootfinding methods. These include Newton, secant, and Halley methods. Our computational results with polynomials of degree up to 30 reveal that for small degree polynomials $B_m^{(k-1)}$ is more efficient than $B_m^{(k)}$, but as the degree increases, $B_m^{(k)}$ becomes more efficient than $B_m^{(k-1)}$. The most efficient of the nine methods is $B_4^{(4)}$, having theoretical order of convergence equal to 1.927. Newton's method which is often viewed as the method of choice is in fact the least efficient method.

Keywords: Polynomial Zeros, Order of Convergence.

AMS Subject Classification. 65H05, 65Y20.

1 Introduction.

For each natural number m greater than one, and each natural number k less than or equal to m, there exists a k-point iteration function for rootfinding, $B_m^{(k)}$, essentially defined as the ratio of two determinants that depend on the first $m-k$ derivatives of the given function. Thus, for each fixed k, there is a family of k-point iteration functions $\{B_m^{(k)}\}_{m=k}^{\infty}$. For simple roots, the order of convergence of $B_m^{(1)}$ is m. The functions $B_2^{(1)}$ and $B_3^{(1)}$ are Newton's and Halley's iteration functions, respectively. For many results regarding these two iteration functions see Pan [11], Scavo and Thoo [12], Traub [10], Ypma [13], Kalantari [7], and Kalantari and Gerlach [8].

Th one-point family, $\{B_m^{(k)}\}_{m=k}^{\infty}$, and its order of convergence is analyzed in [3]. For the special case of square and cube roots the corresponding one-point family was derived in [2] Th general family $B_m^{(k)}$ is derived in [4] and its order of convergence is analyzed in [5]. It is shown that for each fixed m, the order of convergence of $B_m^{(k)}$ monotonically decreases in k, from m to the largest root of the characteristic polynomial of generalized Fibonacci numbers of order m. The following diagram represents the ascending order of convergence of these members and their corresponding orders:

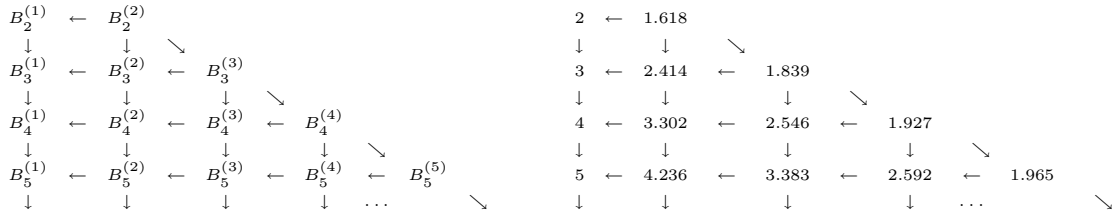

Figure 1: The ascending order and partial list of actual orders.

In this paper we will make a computational comparison of the first nine members in finding roots of the polynomial

$$f(x) = c_t x^t + c_{t-1} x^{t-1} + \ldots + c_0,$$

where c_i's are reals. The iteration functions can also be applied in the complex plane. However, in this paper we will restrict ourselves to experimentation with finding real roots of polynomials. The family of iteration functions $B_m^{(k)}$ has tremendous utility and significance in rootfinding. For some recent results see [6, 7, 8]. For a large bibliography on the rootfinding problem see McNamee [9], and for a recent survey article on some polynomial rootfinding methods see Pan [11].

In Section 2, we formally define the first nine members of $B_m^{(k)}$. In Section 3, we derive their iteration complexity. In Section 4, we describe our experimental results, and in Section 5, our conclusion.

2 The iteration functions.

Definition 1. A vector $a = (x_1, \ldots, x_{n+1}) \in \Re^{n+1}$ is said to be an *admissible vector of nodes*, if whenever $x_i = x_j$, $i < j$, we have $x_i = x_{i+1} = \cdots = x_j$. If the number of distinct x_i's is k, we shall say a is *k-point admissible*. In the special case of $k = 1$, we identify a with the common value, x_1. We shall say a is *monotonic k-point*, if it is k-point admissible of the type $a = (x_1, \ldots, x_1, x_2, \ldots, x_k)$, where $x_i \neq x_j$, if $i \neq j$. Let $a = (x_1, \ldots, x_{n+1})$ be an admissible vector of nodes. For any pair of indices i, j satisfying $1 \leq i \leq j \leq (n+1)$, inductively define the *confluent divided differences* as

$$f_{ij} = \begin{cases} \frac{f^{(j-i)}(x_i)}{(j-i)!}, & \text{if } x_i = x_j \\ \frac{f_{i+1,j} - f_{i,j-1}}{(x_j - x_i)}, & \text{otherwise.} \end{cases}.$$

We now define the iteration functions for $m = 2, 3, 4$, and $k \leq m$:

$$B_2^{(k)}(a_2) = x_1 - f_{11}\frac{1}{f_{12}}, \quad B_3^{(k)}(a_3) = x_1 - f_{11}\,\frac{f_{23}}{\begin{vmatrix} f_{12} & f_{13} \\ f_{22} & f_{23} \end{vmatrix}}, \quad B_4^{(k)}(a_4) = x_1 - f_{11}\,\frac{\begin{vmatrix} f_{23} & f_{24} \\ f_{33} & f_{34} \end{vmatrix}}{\begin{vmatrix} f_{12} & f_{13} & f_{14} \\ f_{22} & f_{23} & f_{24} \\ 0 & f_{33} & f_{34} \end{vmatrix}}.$$

Let θ be a simple root of f. It can be shown, see [4], that for each $k = 1, \ldots, m$, there exists a neighborhood of θ such that given any initial k-point monotonic vector

$$a_m^{(0)} = (x_1^{(0)}, \ldots, x_1^{(0)}, x_2^{(0)}, \ldots, x_k^{(0)}) \in \Re^m,$$

the fixed-point iteration that for each $r \geq 0$ replaces $a_m^{(r)}$ with the monotonic k-point vector

$$a_m^{(r+1)} = \left(B_m^{(k)}(a_m^{(r)}), \ldots, B_m^{(k)}(a_m^{(r)}), x_1^{(r)}, \ldots, x_{k-1}^{(r)} \right) \in \Re^m,$$

is well-defined. Moreover, the sequence of points $\{x_1^{(r)}\}_{r=0}^{\infty}$ converges to θ having order of convergence as given in Figure 1.

3 The iteration complexity of the methods.

Here we will analyze the iteration complexity of the nine rootfinding methods of interest for the case of polynomials. More precisely, we are interested in the number of arithmetic operation, $(+,-,\times,\div)$, within a given number of iterations of $B_m^{(k)}$, as applied to a given polynomial.

Let $i_m^{(k)}$ be the number of iterations of $B_m^{(k)}$ as applied to a given polynomial. If $k = 1$, in order to go from one iteration to the next, we would have to recompute new function-derivative values. However, for $k > 1$, to go from one iteration to the next it is possible to make use of some precomputed values. Thus, after the initial iteration the complexity of evaluation of $B_m^{(k)}$ could improve. Let $N_{m,j}^{(k)}$ denote the number of arithmetic operation needed to compute $B_m^{(k)}$ in the j-th iteration. Clearly, we have

$$N_{m,1}^{(k)} \geq N_{m,2}^{(k)}, \quad N_{m,2}^{(k)} = N_{m,j}^{(k)},$$

for all $j > 1$. Let $N_m^{(k)}$ be the number of arithmetic operations performed after $i_m^{(k)}$ iterations of $B_m^{(k)}$. Then, we have

$$N_m^{(k)} = N_{m,1}^{(k)} + (i_m^{(k)} - 1)N_{m,2}^{(k)}.$$

The above equation can be written in terms of $i_m^{(k)}$, and the quantity $T(f^{(j)})$, defined as the number of arithmetic operations needed to compute the j-th derivative of the given polynomial.

As an example, consider $B_4^{(4)}(a_4)$, where $a_4 = (x_1, x_2, x_3, x_4)$, $x_i \neq x_j$. We have

$$B_4^{(4)}(a_4) = x_1 - \frac{f_{11}(f_{23}f_{34} - f_{33}f_{24})}{f_{12}(f_{23}f_{34} - f_{33}f_{24}) - f_{22}(f_{13}f_{34} - f_{33}f_{14})}.$$

It is easy to see that having computed the divided differences we need 12 arithmetic operations to compute $B_4^{(4)}(a_4)$. The computation of the divided differences can be represented by the following diagram:

$$\begin{array}{ccccccccc}
f(x_1) & & & & & & & & \\
\mid & \searrow & & & & & & & \\
f(x_2) & \rightarrow & f_{12} & & & & & & \\
\mid & \searrow & & \searrow & & & & & \\
f(x_3) & \rightarrow & f_{23} & \rightarrow & f_{13} & & & & \\
\mid & \searrow & & \searrow & & \searrow & & & \\
f(x_4) & \rightarrow & f_{34} & \rightarrow & f_{24} & \rightarrow & f_{14} & &
\end{array}$$

Figure 2: The divided differences needed for the first iteration of $B_4^{(4)}$.

Since x_i's are distinct, the computation of each f_{ij} requires three arithmetic operations. Thus, $N_{4,1}^{(4)} = 4T(f) + 6 \times 3 + 12 = 4T(f) + 30$. In the next iteration the input (x_1, x_2, x_3, x_4) is

replaced with $(B_4^{(4)}(a_4), x_1, x_2, x_3)$. Thus, we need to compute the new divided differences:

$$
\begin{array}{ccccccccc}
f(B_4^{(4)}(a_4)) & & & & & & & & \\
\quad | & \searrow & & & & & & & \\
f(x_1) & \rightarrow & f_{12} & & & & & & \\
\quad | & \searrow & & \searrow & & & & & \\
f(x_2) & \rightarrow & f_{23} & \rightarrow & f_{13} & & & & \\
\quad | & \searrow & & \searrow & & \searrow & & & \\
f(x_3) & \rightarrow & f_{34} & \rightarrow & f_{24} & \rightarrow & f_{14} &
\end{array}
$$

Figure 3: The divided differences needed for the second iteration of $B_4^{(4)}$.

However, we note that the only new calculation is the computation of $f(B_4^{(4)}(a_4))$, f_{12}, f_{13}, and f_{14}. Thus, $N_{4,2}^{(4)} = T(f) + 3 \times 3 + 12 = T(f) + 21$. Therefore,

$$N_4^{(4)} = 4T(f) + 30 + (i_4^{(4)} - 1)(T(f) + 21).$$

Likewise, we can compute $N_m^{(k)}$ for all the nine iteration functions. This would require the computational complexity of the first and second iterations, taking into account the computation of the corresponding confluent divided differences. It can be show that this gives:

$$N_2^{(k)} = \begin{cases} i_2^{(1)}[T(f) + T(f') + 2], & \text{if } k = 1 \\ i_2^{(2)}[T(f) + 5] + T(f), & \text{if } k = 2. \end{cases}$$

$$N_3^{(k)} = \begin{cases} i_3^{(1)}[T(f) + T(f') + T(f'') + 7], & \text{if } k = 1 \\ i_3^{(2)}[T(f) + T(f') + 11] + T(f), & \text{if } k = 2 \\ i_3^{(3)}[T(f) + 12] + 2T(f) + 3, & \text{if } k = 3. \end{cases}$$

$$N_4^{(k)} = \begin{cases} i_4^{(1)}[T(f) + T(f') + T(f'') + T(f''') + 14], & \text{if } k = 1 \\ i_4^{(2)}[T(f) + T(f') + T(f'') + 20] + T(f), & \text{if } k = 2 \\ i_4^{(3)}[T(f) + T(f') + 22] + 2T(f) + 3, & \text{if } k = 3 \\ i_4^{(4)}[T(f) + 21] + 3T(f) + 9, & \text{if } k = 4. \end{cases}$$

Figure 4: The iteration complexity of $B_m^{(k)}$ after $i_m^{(k)}$ iterations.

4 The experiment.

In this section we describe our experimentation with $B_m^{(k)}$ in finding real roots of polynomials. Specifically, the following guidelines were used.

For each degree t in the range $[2, 30]$, we generated 10 random polynomials in the following fashion:

The coefficient of the highest degree was a randomly chosen integer in the interval $[1, 10]$, and the constant term a randomly chosen integer in $[-100, -1]$. Thus, each generated polynomial had a positive root. All other coefficients were random integers in the interval $[-10, 10]$. All random numbers were generated using uniform distribution.

For each random polynomial, 15 random starting inputs were generated. In the case of multipoint iteration functions, say $k = 4$, we would augment each input, say x_1, to a vector of inputs, (x_1, x_2, x_3, x_4), where $x_i - x_{i-1}$ was chosen to be the same random integer between 2 and 6. Then, given each starting input we applied the nine iteration functions. The evaluation of a given polynomial and its derivatives were carried out using Horner's method.

For a given polynomial we applied each of the nine iteration function until the following stopping criterion was satisfied,

$$|B_m^{(k)}(a_m^{(r)}) - B_m^{(k)}(a_m^{(r-1)})| < 10^{-15}.$$

For each of the nine iteration functions we kept track of the total number of arithmetic operations needed to approximate the same root, i.e., the number $N_m^{(k)}$, until the above stopping criterion was satisfied. In case the iterates of any of the nine iteration function did not converge, or they did not all converge to the same root, we would generate another random input, and in some cases even a new random polynomial. This ensured that we would have a fair comparison of the iteration functions. For each given polynomial, we computed the ratio

$$R_m^{(k)} = \frac{N_m^{(k)}}{N_2^{(1)}},$$

for each of the 15 inputs. This ratio can be viewed as relative efficiency of $B_m^{(k)}$ with respect to Newton's method, i.e., $B_2^{(1)}$.

For example for the polynomial with coefficients$(c_3, c_2, c_1, c_0) = (1, -5, 6, -30)$, the first input vector was $a_4 = (75, 77, 79, 81)$ and we obtained

$$i_m^{(k)}: \begin{array}{ccccccc} 14 & \leftarrow & 19 & & & & \\ \downarrow & & \downarrow & \searrow & & & \\ 9 & \leftarrow & 10 & \leftarrow & 14 & & \\ \downarrow & & \downarrow & & \downarrow & \searrow & \\ 7 & \leftarrow & 8 & \leftarrow & 9 & \leftarrow & 12 \end{array} \qquad N_m^{(k)}: \begin{array}{ccccccc} 154 & \leftarrow & 195 & & & & \\ \downarrow & & \downarrow & \searrow & & & \\ 162 & \leftarrow & 205 & \leftarrow & 251 & & \\ \downarrow & & \downarrow & & \downarrow & \searrow & \\ 175 & \leftarrow & 253 & \leftarrow & 292 & \leftarrow & 336 \end{array}$$

$$R_m^{(k)}: \begin{array}{ccccccc} 1.00 & \leftarrow & 1.27 & & & & \\ \downarrow & & \downarrow & \searrow & & & \\ 1.05 & \leftarrow & 1.33 & \leftarrow & 1.63 & & \\ \downarrow & & \downarrow & & \downarrow & \searrow & \\ 1.14 & \leftarrow & 1.64 & \leftarrow & 1.90 & \leftarrow & 2.18 \end{array}$$

We now summarize our computational results based on polynomials with degree in the range $[2, 30]$. For a given polynomial the quantity $R_m^{(k)}$ (defined above) was computed. This ratio

was then averaged over the 15 random inputs. Finally, this average was averaged over the 10 randomly generated polynomials of a given degree. This final ratio gives the average relative efficiency of $B_m^{(k)}$ with respect to Newton's method. The following figure graphs this average efficiency as a function of degree. In order to get a better picture, we have enlarged this graph at degrees 9, 16, and 23.

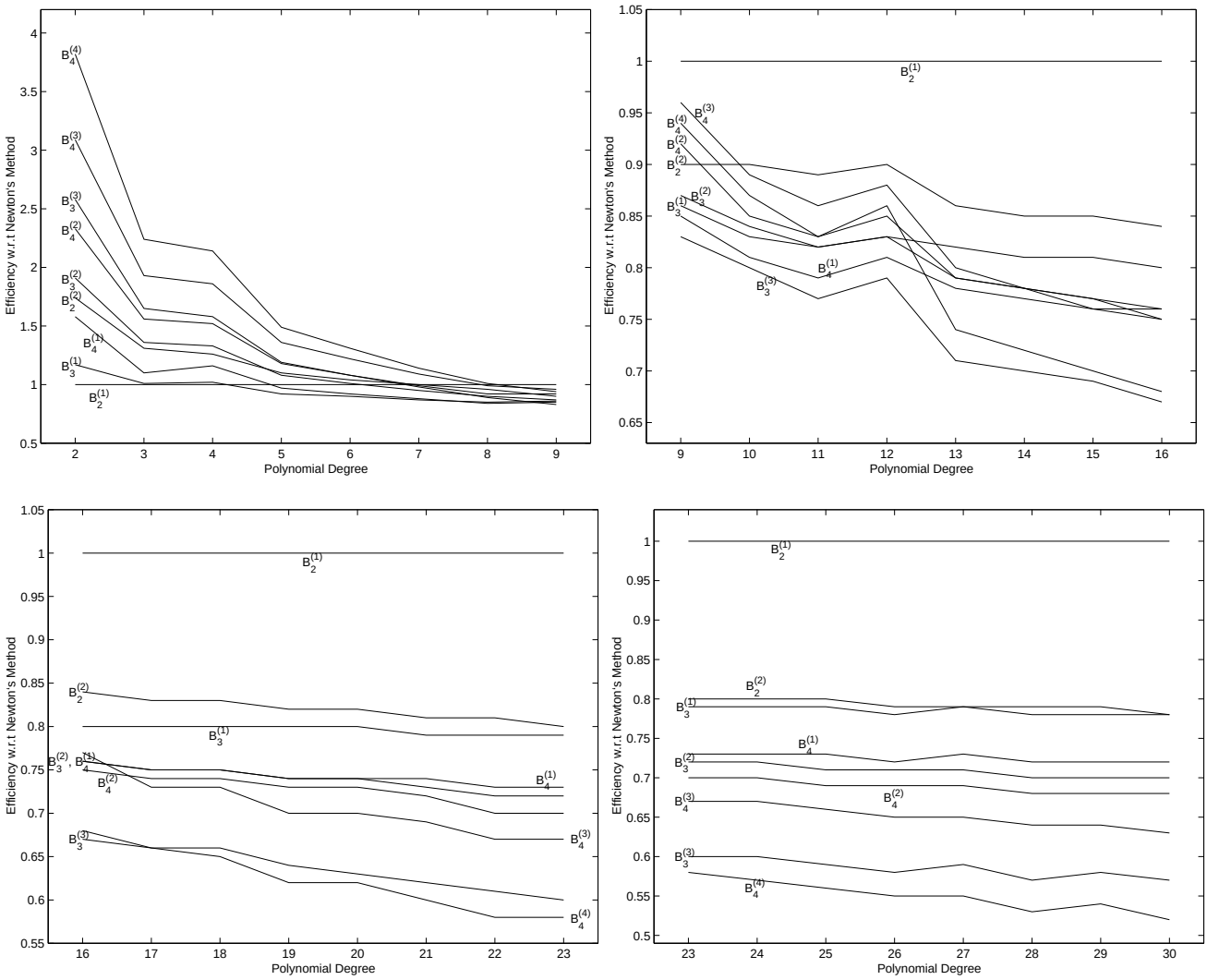

Figure 5: Average efficiency ratio of B_m^k versus Newton's method.

5 Conclusion.

Based on our experimentation we see that for very small degrees Newton's method is the best. But, as the degree of polynomial increases, in comparison with other methods, Newton's method becomes less and less effective. Also, this experiment shows that, for very small degrees, $B_m^{(k-1)}$ is better than $B_m^{(k)}$, but, as the degree increases, $B_m^{(k)}$ becomes more efficient than $B_m^{(k-1)}$. As the degree increases, $B_4^{(4)}$ becomes the most efficient among the nine methods. This method which requires no derivative evaluations, has theoretical order of convergence very close to Newton's quadratic order. We expect that for larger values of m than $m = 4$, and for larger degree polynomials, one would observe the same behavior. Finally, one would expect that the same results would apply to the computation of complex roots of polynomials.

References

[1] B. Kalantari, and I. Kalantari, *High order iterative methods for approximating square roots*, BIT 36 (1996), pp. 395-399.

[2] B. Kalantari, I. Kalantari, and R. Zaare-Nahandi, *A basic family of iteration functions for polynomial root finding and its characterizations*, J. of Comp. and Appl. Math., 80 (1997), pp. 209-226.

[3] B. Kalantari, *Generalization of Taylor's theorem and Newton's method via a new family of determinantal interpolation formulas*, Tech. Report DCS-TR 328, Dept. of Computer Science, Rutgers University, New Brunswick, NJ, 1997.

[4] B. Kalantari, *On the order of a determinantal family of root-finding methods*, Tech. Report DCS-TR 329, Dept. of Computer Science, Rutgers University, New Brunswick, NJ, 1997. To appear in *BIT*.

[5] B. Kalantari, *Approximation of polynomial root using a single input and the corresponding derivative values*, Tech. Report DCS-TR 369, Dept. of Computer Science, Rutgers University, New Brunswick, NJ, 1998.

[6] B. Kalantari, *Halley's method as the first member of an infinite family of cubic order rootfinding methods*, Tech Report DCS-TR 370, Dept. of Computer Science, Rutgers University, New Brunswick, NJ, 1998.

[7] B. Kalantari and J. Gerlach, *Newton's method and generation of a determinantal family of iteration functions*, Tech. Report DCS-TR 371, Dept. of Computer Science, Rutgers University, New Brunswick, NJ, 1998.

[8] J.M. McNamee, *A bibliography on root of polynomials*, J. of Comp. and Appl. Math., 47 (1993), pp. 391-394.

[9] J. F. Traub, *Iterative Methods for the Solution of Equations*, New Jersey, Prentice Hall, 1964.

[10] V. Y. Pan, *Solving a polynomial equation: some history and recent progress*, SIAM Review, 39 (1997), pp. 187–220.

[11] T. R. Scavo, and J.B. Thoo, *On the geometry of Halley's method*, Amer. Math. Monthly, 102 (1995), pp. 417-426.

[12] T. J. Ypma, *Historical development of Newton-Raphson method*, SIAM Review, 37 (1995), pp. 531–551.