Dynamical control of Newton's method for multiple roots of polynomials

Stef Graillat, Fabienne Jézéquel, Moustadrani Saïd Ibrahim

Dynamical control of Newton's method for multiple roots of polynomials*

S. Graillat[1], F. Jézéquel[1,2], and M. S. Ibrahim

[1]Sorbonne Universités, UPMC Univ Paris 06, CNRS, UMR 7606, Laboratoire d'Informatique de Paris 6 (LIP6), 4 place Jussieu, 75252 Paris CEDEX 05, France
[2]Université Panthéon-Assas, 12 place du Panthéon, 75231 Paris CEDEX 05, France
{Stef.Graillat,Fabienne.Jezequel}@lip6.fr

Abstract

In this article, we show how to perform a dynamical control of Newton's method for the computation of multiple roots of polynomials. Using Discrete Stochastic Arithmetic, root approximations are computed until the difference between two successive approximations is numerical noise. With such a stopping criterion, the optimal number of iterations in Newton's method are performed. Moreover it is possible to estimate in the result obtained which digits are in common with the exact root. Two strategies to estimate the multiplicity of polynomials roots are compared: one requires root approximations computed at different precisions and the other three successive iterates of Newton's method. We show that using such a strategy and then the modified Newton's method, multiple roots can be computed with a requested accuracy.

Keywords: Discrete Stochastic Arithmetic, floating-point arithmetic, numerical validation, rounding errors, polynomial, multiple roots, Newton's method, modified Newton's method

AMS subject classifications: 65H10, 65G20, 65H05, 65-04

1 Introduction

The approximation of polynomial roots is usually based on a sequence computation and it might be difficult to perform the optimal number of iterations. Indeed if the computation stops too early, the result may be still far from the exact root and too many iterations may cause an increase of the rounding error that affects the result. In this article we show how to perform a dynamical control of Newton's method for the approximation of a root α of multiplicity m of a polynomial f, *i.e.* such that $f^{(i)}(\alpha) = 0$ for $i = 0, \ldots, m-1$ and $f^{(m)}(\alpha) \neq 0$. Thanks to Discrete Stochastic Arithmetic [29], a stopping criterion that takes into account rounding errors can be used and the optimal number of iterates can be computed. Moreover, from the root multiplicity m, it is possible to estimate the number of correct digits in the approximation obtained.

In this article, two strategies are compared to estimate the multiplicity of polynomial roots. It is well-known that deciding whether a univariate polynomial has a multiple root is an ill-posed problem: an arbitrary small perturbation of a polynomial coefficient may change the answer from yes to no. For example a real double root may be changed into two simple roots. As a consequence, it is hardly possible to verify that

*This work was partly supported by the project FastRelax ANR-14-CE25-0018-01.

a polynomial has a multiple root, unless the entire computation is performed without any rounding errors, that is to say using Computer Algebra methods. In [23], we showed that it is possible to certify with interval techniques that the polynomial f has a true multiple root in its neighborhood. Given an integer m and an approximation x_0 of the root α, we provide a method that can potentially find intervals X and E such that we can certify that there exist $\widehat{\alpha} \in X$ and $e \in E$, $\widehat{\alpha}$ being a root of multiplicity m of $g(x) := f(x) - e$. If the polynomial f has a root of multiplicity m, typically the interval E is a very narrow interval around zero. Of course this method can fail and answer that we cannot say anything. But if this method provides some intervals then this is a true (mathematical) solution. In this paper, we describe how to find a number m such that the previous procedure works, meaning that there is a polynomial close to f that has a true root of multiplicity m.

This paper is organized as follows. In Section 2, we describe how rounding errors can be estimated using Discrete Stochastic Arithmetic. In Section 3, we propose a method and some theoretical results to dynamically control the errors in Newton's method. In Section 4, we present how to estimate the multiplicity of polynomial roots thanks to a dynamical control of Newton's method with Discrete Stochastic Arithmetic. Finally, concluding remarks and perspectives are presented in Section 5.

2 Estimation of rounding errors using Discrete Stochastic Arithmetic

2.1 Principles of the CESTAC method

Based on a probabilistic approach, the CESTAC method [28] allows the estimation of rounding error propagation which occurs with floating-point arithmetic. When no overflow occurs, the exact result, r, of any non exact floating-point arithmetic operation is bounded by two consecutive floating-point values R^- and R^+. The basic idea of the method is to perform each arithmetic operation N times, randomly rounding each time, with a probability of 0.5, to R^- or R^+. The computer's deterministic arithmetic, therefore, is replaced by a stochastic arithmetic where each arithmetic operation is performed N times before the next one is executed, thereby propagating the rounding error differently each time.

It has been proved [4] that the computed result R of n elementary arithmetic operations is modelled to the first order in 2^{-p} as:

$$R \approx Z = r + \sum_{i=1}^{n} g_i(d) 2^{-p} z_i, \tag{1}$$

where r is the exact result, $g_i(d)$ are coefficients depending exclusively on the data and on the code, p is the number of bits in the mantissa and z_i are independent uniformly distributed random variables on $[-1, 1]$.

From Equation 1, we deduce that:

1. the mean value of the random variable Z is the exact result r;
2. the distribution of Z is a quasi-Gaussian distribution.

Then by identifying R and Z, *i.e.* by neglecting all the second order terms, Student's test can be used to estimate the accuracy of R. From N samples R_i, $i = 1, 2, ..., N$,

$$\forall \beta \in [0,1], \exists \tau_\beta \in \mathbb{R} \text{ s.t. } P\left(\left| \overline{R} - r \right| \leq \frac{\sigma \tau_\beta}{\sqrt{N}} \right) = 1 - \beta, \tag{2}$$

where

$$\overline{R} = \frac{1}{N} \sum_{i=1}^{N} R_i \quad \text{and} \quad \sigma^2 = \frac{1}{N-1} \sum_{i=1}^{N} \left(R_i - \overline{R} \right)^2. \tag{3}$$

τ_β is the value of Student's distribution for $N-1$ degrees of freedom and a probability level $1-\beta$. Therefore, if no overflow occurs, the number of decimal significant digits common to $\overline{R}$ and r can be estimated as

$$C_{\overline{R}} = \log_{10}\left(\frac{\sqrt{N}\,|\overline{R}|}{\sigma\tau_\beta}\right). \tag{4}$$

Thus the implementation of the CESTAC method in a code providing a result R consists in:

- performing N times this code with the random rounding mode, which is obtained by using randomly the upward or downward rounding mode; we then obtain N samples R_i of R;
- choosing as the computed result the mean value $\overline{R}$ of R_i, $i = 1, ..., N$;
- estimating with Equation 4 the number of exact decimal significant digits of $\overline{R}$.

In practice $\beta = 0.05$ and $N = 3$. Indeed, it has been shown [4, 5] that $N = 3$ is in some reasonable sense the optimal value. The estimation with $N = 3$ is more reliable than with $N = 2$ and increasing the size of the sample does not improve the quality of the estimation. The probability of overestimating the number of exact significant digits of at least one is 0.054% and the probability of underestimating the number of exact significant digits of at least one is 29%. By choosing $\beta = 0.05$, we prefer to guarantee a minimal number of exact significant digits with a high probability (99.946%), even if we are often pessimistic by one digit. The complete theory can be found in [4, 28].

2.2 Validity of the CESTAC method

Equations 1 and 4 hold if the two following hypotheses are verified:

1. the rounding errors α_i are independent, centered uniformly distributed random variables;
2. the approximation to the first order in 2^{-p} is legitimate.

Concerning the first hypothesis, with the use of the random arithmetic, rounding errors α_i are random variables, however, in practice, they are not rigorously centered and in this case Student's test gives a biased estimation of the computed result. It has been proved [4] that, with a bias of a few σ, the error on the estimation of the number of exact significant digits of $\overline{R}$ is less than one decimal digit. Therefore even if the first hypothesis is not rigorously satisfied, the estimation obtained with Equation 4 is still correct up to one digit.

Concerning the second hypothesis, the approximation to the first order only concerns multiplications and divisions. Indeed the rounding error generated by an addition or a subtraction does not contain any term of higher order. It has been shown [4] that, if both operands in a multiplication or the divisor in a division become insignificant, *i.e.* with no more exact significant digit, then the first order approximation may be not legitimate. In practice, the CESTAC method requires, during the execution of the code, a dynamical control of multiplications and divisions, which is a so-called *self-validation* of the method.

2.3 Principles of DSA

The self-validation of the CESTAC method requires its synchronous implementation. Indeed to enable the estimation of the accuracy, the samples which represent a result must be computed simultaneously. Discrete Stochastic Arithmetic (DSA) [29] has been defined from the synchronous implementation of the CESTAC method. With DSA, a real number becomes an N-dimensional set and any operation on these N-dimensional sets is performed element per element using the random rounding mode. The number of exact significant digits of such an N-dimensional set can be estimated from Equation 4. The self-validation of the CESTAC method also leads to the concept of computational zero [27] defined below.

Definition 2.1 *During the run of a code using the CESTAC method, a result R is a computational zero, denoted by @.0, if $\forall i, R_i = 0$ or $C_{\overline{R}} \leq 0$.*

Any computed result R is a computational zero if either $R = 0$, R being significant, or R is insignificant. A computational zero is a value that cannot be differentiated from the mathematical zero because of its rounding error. From the concept of computational zero, an equality concept and order relations have been defined for DSA.

Definition 2.2 *Let X and Y be N-samples provided by the CESTAC method.*

- *Discrete stochastic equality denoted by $ds=$ is defined as $X ds= Y$ if and only if $X - Y = @.0$.*
- *Discrete stochastic inequalities denoted by $ds>$ and $ds\geq$ are defined as:*
  $X ds> Y$ if and only if $\overline{X} > \overline{Y}$ and $X ds\neq Y$,
  $X ds\geq Y$ if and only if $\overline{X} \geq \overline{Y}$ or $X ds= Y$.

Stochastic relational operators ensure that in a branching statement the same sequence of instructions is performed for all the samples which represent a variable. DSA enables to estimate the impact of rounding errors on any result of a scientific code and also to check that no anomaly occurred during the run, especially in branching statements.

2.4 Implementation of DSA

The CADNA[1] software [4, 7, 10, 11, 19] is a library which implements DSA in any code written in C, C++ or Fortran and allows one to use new numerical types: the stochastic types. In essence, classical floating-point variables (in single or in double precision) are replaced by the corresponding stochastic variables, which are composed of three perturbed floating-point values. The library contains the definition of all arithmetic operations and order relations for the stochastic types. The control of the accuracy is performed only on variables of stochastic type. When a stochastic variable is printed, only its exact significant digits appear. For a computational zero, the string "@.0" is printed. In contrast to interval arithmetic, that computes guaranteed results, the CADNA software provides, with the probability 95% the number of exact significant digits of any computed result. CADNA has been successfully used for the numerical validation of academic and industrial simulation codes in various domains such as astrophysics [14], atomic physics [24], chemistry, climate science [3, 12], fluid dynamics, geophysics [13].

The SAM library[2] [9] implements in arbitrary precision the features of DSA: the stochastic types, the concept of computational zero and the stochastic operators. The SAM library is written in C++ and is based on MPFR. The particularity of SAM (compared to CADNA) is the arbitrary precision of stochastic variables. The SAM library with 24-bit (resp. 53-bit) mantissa length is similar to CADNA in single (resp. double) precision, except the range of the exponent is only limited by the machine memory. In SAM, the number of exact significant digits of any stochastic variable is estimated with the probability 95%, whatever its precision. Like in CADNA, the arithmetic and relational operators in SAM take into account rounding error propagation.

In CADNA and SAM all operators are overloaded, therefore their use in a program requires only a few modifications: essentially changes in the declarations of variables and in input/output statements. CADNA and SAM can detect numerical instabilities which occur during the execution of the code. These instabilities are either operations involving a computational zero or cancellations, *i.e.* subtractions of two very close values which generates a sudden loss of accuracy. At the end of the run, each type of instability together with their occurrences are printed.

3 Dynamical control of Newton's method

In this section, we will first recall Newton's method and some results on its convergence. We will then state a criterion to stop the iterations and give some information about the accuracy of the computed solution. Numerical experiments will confirm our results.

[1] `http://cadna.lip6.fr`

[2] `http://www-pequan.lip6.fr/~jezequel/SAM`

3.1 Newton's method

Newton's method enables one to approximate a root α of a function f by computing the sequence $x_{n+1} = g(x_n)$ with $g(x) = x - \frac{f(x)}{f'(x)}$. Newton's method is based on the fixed-point method; indeed $g(x) = x \iff f(x) = 0$. If α is a simple root ($f'(\alpha) \neq 0$), then there is a local quadratic convergence. If now α is a multiple root of multiplicity $m > 1$ then the convergence is linear. If one knows the multiplicity, it is possible to get a quadratic convergence via the modified Newton's method defined by the iteration $x_{n+1} = x_n - m\frac{f(x_n)}{f'(x_n)}$. One can find more information on Newton's method in the book [6]. For some results on the convergence and the accuracy of Newton's method in floating-point arithmetic, see [8, 15, 25].

3.2 Optimal stopping criterion

The following definition makes clear the notion of decimal significant digits in common between two numbers.

Definition 3.1 *The number of decimal significant digits in common between two real numbers a and b is defined in $\boldsymbol{R}$ by*

- *for $a \neq b$, $C_{a,b} = \log_{10}\left|\dfrac{a+b}{2(a-b)}\right|$;*
- *for all $a \in \boldsymbol{R}$, $C_{a,a} = +\infty$.*

Then $|a-b| = \left|\frac{a+b}{2}\right| 10^{-C_{a,b}}$. For instance, if $C_{a,b} = 3$, the relative difference between a and b is of the order of 10^{-3}, which means that a and b have three significant decimal digits in common.

Remark 3.1 *The value of $C_{a,b}$ can seem surprising if we consider the decimal notations of a and b. For example, if $a = 2.4599976$ and $b = 2.4600012$, then $C_{a,b} \approx 5.8$. The difference due to the sequences of "0" or "9" is illusive. The significant decimal digits of R and r become actually different from the sixth position.*

The following lemma gives a relation between the derivative of the function g at the root α and the multiplicity.

Lemma 3.1 *Let us assume α is a root of multiplicity $m \geq 2$ of a polynomial f and $g(x) = x - \frac{f(x)}{f'(x)}$. Then $g'(\alpha) = 1 - \frac{1}{m}$.*

Proof: Let us assume that α is a root of f of multiplicity m, then we can find a function h such that

$$\forall x \in I,\ f(x) = (x-\alpha)^m h(x) \text{ with } h(\alpha) \neq 0, \tag{5}$$

with I a small open interval around α. As a consequence, it turns out that

$$f'(x) = (x-\alpha)^{m-1}\left[mh(x) + (x-\alpha)h'(x)\right]. \tag{6}$$

Then

$$f''(x) = (x-\alpha)^{m-1}\left[(m+1)h'(x) + (x-\alpha)h''(x)\right] + (m-1)(x-\alpha)^{m-2}\left[mh(x) + (x-\alpha)h'(x)\right]. \tag{7}$$

The derivative of g is

$$g'(x) = 1 - \frac{(f'(x))^2 - f(x)f''(x)}{(f'(x))^2} = \frac{f(x)f''(x)}{(f'(x))^2}. \tag{8}$$

Therefore we have

$$g'(x) = \frac{(x-\alpha)h(x)\left[(m+1)h'(x) + (x-\alpha)h''(x)\right]}{\left[mh(x) + (x-\alpha)h'(x)\right]^2} + \frac{(m-1)h(x)}{mh(x) + (x-\alpha)h'(x)}, \tag{9}$$

which implies that

$$g'(\alpha) = 1 - \frac{1}{m}. \tag{10}$$

□

In the next two lemmas, we recall well-known results on finite expansions that can be found in any textbook on mathematical analysis. They will be used in the sequel.

Lemma 3.2 *Let us consider (u_n) and (v_n) two real sequences ($n \in \boldsymbol{N}$). Then we have*

$$\lim_{n\to\infty} \frac{u_n}{v_n} = 1 \iff u_n \sim_\infty v_n.$$

Lemma 3.3 *Let us consider (u_n) and (v_n) two real sequences such that $\forall n \in \boldsymbol{N}, u_n > 0$ and $v_n > 0$, $\lim_{n\to\infty} u_n = L \neq 1$ and $u_n \sim_\infty v_n$. Then $\log u_n \sim_\infty \log v_n$.*

The following theorem gives a relation between the common significant digits of two successive approximations of the root and the common significant digits of an approximation and the root when the root is *simple*.

Theorem 3.1 *Let x_n and x_{n+1} be two successive approximations computed using Newton's method of a polynomial root α of multiplicity $m = 1$ (simple root).*

Then

$$C_{x_n,x_{n+1}} \sim_\infty C_{x_n,\alpha}.$$

Proof: If $n \in \boldsymbol{N}$ exists such that $x_n = x_{n+1}$. Then $x_{n+1} = \alpha$, $C_{x_n,\alpha} = \infty$, and $C_{x_n,x_{n+1}} = \infty$. So, in this case, Theorem 3.1 is satisfied.

Let us now consider $\forall n \in \boldsymbol{N}, x_n \neq x_{n+1}$. From Taylor expansion, $f(x_n) = f(\alpha) + f'(\lambda_n)(x_n - \alpha)$ where $\lambda_n \in [\min(\alpha, x_n), \max(\alpha, x_n)]$. Moreover, with Newton iteration, we have $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$. As a consequence, $x_{n+1} = x_n + M_n(\alpha - x_n)$ with $M_n = f'(\lambda_n)/f'(x_n)$ and $M_n \to 1$ when $n \to +\infty$. It turns out that

$$\lim_{n\to+\infty} \frac{(\alpha - x_n)(x_{n+1} + x_n)}{(x_{n+1} - x_n)(\alpha + x_n)} = 1.$$

Applying Lemma 3.2, we obtain

$$\frac{x_{n+1} + x_n}{x_{n+1} - x_n} \sim_\infty \frac{\alpha + x_n}{\alpha - x_n}. \tag{11}$$

Therefore, we have

$$\left|\frac{x_{n+1} + x_n}{2(x_{n+1} - x_n)}\right| \sim_\infty \left|\frac{\alpha + x_n}{2(\alpha - x_n)}\right|. \tag{12}$$

Applying Lemma 3.3, we also obtain

$$\log_{10}\left|\frac{x_{n+1} + x_n}{2(x_{n+1} - x_n)}\right| \sim_\infty \log_{10}\left|\frac{\alpha + x_n}{2(\alpha - x_n)}\right|. \tag{13}$$

Finally, from Definition 3.1, it turns out that

$$C_{x_n,x_{n+1}} \sim_\infty C_{x_n,\alpha}. \tag{14}$$

□

From Theorem 3.1, if the convergence zone is reached, then the digits common to two successive approximations x_n and x_{n+1} are also in common with the exact root α. If iterations are performed until the difference between two successive approximations x_n and x_{n+1} is not significant, then the transformation

of x_n into x_{n+1} is due to rounding errors. Further iterations are useless: the number of iterations has been optimized. Moreover, if the difference $x_n - x_{n+1}$ is not significant, the digits of x_{n+1} which are not affected by rounding errors are in common with x_n. Therefore, from Theorem 3.1, the decimal digits of the approximation x_n which are not affected by rounding errors are also in common with the exact root α.

The following theorem gives a relation between the common significant digits of two successive approximations of the root and the common significant digits of an approximation and the root when the root is *multiple*.

Theorem 3.2 *Let x_n and x_{n+1} be two successive approximations computed using Newton's method of a polynomial root α of multiplicity $m \geq 2$.*

Then

$$C_{x_n,x_{n+1}} \sim_\infty C_{x_{n+1},\alpha} + \log_{10}(m-1).$$

Proof: If $n \in \boldsymbol{N}$ exists such that $x_n = x_{n+1}$. Then $x_{n+1} = \alpha$, $C_{x_{n+1},\alpha} = \infty$ and $C_{x_n,x_{n+1}} = \infty$. So, in this case, Theorem 3.2 is satisfied.

Let us now consider $\forall n \in \boldsymbol{N}, x_n \neq x_{n+1}$. Applying Taylor-Lagrange formula, it exists $\lambda_n \in [\min(\alpha, x_n), \max(\alpha, x_n)]$ such that

$$g(\alpha) - g(x_n) = g'(\lambda_n)(\alpha - x_n). \tag{15}$$

As $\lim_{n\to\infty} x_n = \alpha$ and $\lim_{n\to\infty} g'(\lambda_n) = g'(\alpha) = 1 - \frac{1}{m}$ with $m \geq 2$ from Lemma 3.1, we can assume that $g'(\lambda_n) \neq 0$ and $g'(\lambda_n) \neq 1$.

By definition, we have

$$g(\alpha) - g(x_n) = \alpha - x_{n+1}. \tag{16}$$

By combining Equations 15 and 16 we can write

$$\alpha - x_n = \frac{\alpha - x_{n+1}}{g'(\lambda_n)}. \tag{17}$$

Therefore, we have

$$\alpha - x_{n+1} = \frac{\alpha - x_{n+1}}{g'(\lambda_n)} + x_n - x_{n+1}. \tag{18}$$

Because $x_n \neq x_{n+1}$, from Equation 18, we can write

$$\frac{\alpha - x_{n+1}}{x_n - x_{n+1}} = \frac{g'(\lambda_n)}{g'(\lambda_n) - 1}. \tag{19}$$

Because $\lim_{n\to\infty} \lambda_n = \alpha$, from Lemma 3.1 and Equation 19, we deduce

$$\lim_{n\to\infty} \frac{\alpha - x_{n+1}}{(1-m)(x_n - x_{n+1})} = 1. \tag{20}$$

Moreover, because $\lim_{n\to\infty} x_n = \alpha$, it follows that

$$\lim_{n\to\infty} \frac{(\alpha - x_{n+1})(x_n + x_{n+1})}{(1-m)(x_n - x_{n+1})(\alpha + x_{n+1})} = 1. \tag{21}$$

Applying Lemma 3.2, we obtain

$$\frac{x_n + x_{n+1}}{x_n - x_{n+1}} \sim_\infty \frac{(1-m)(\alpha + x_{n+1})}{\alpha - x_{n+1}}. \tag{22}$$

and so

$$\left|\frac{x_n + x_{n+1}}{2(x_n - x_{n+1})}\right| \sim_\infty (m-1)\left|\frac{\alpha + x_{n+1}}{2(\alpha - x_{n+1})}\right|. \tag{23}$$

Applying Lemma 3.3, we obtain

$$\log_{10}\left|\frac{x_n + x_{n+1}}{2(x_n - x_{n+1})}\right| \sim_\infty \log_{10}\left|\frac{\alpha + x_{n+1}}{2(\alpha - x_{n+1})}\right| + \log_{10}(m-1). \tag{24}$$

Finally, from Definition 3.1, we deduce

$$C_{x_n,x_{n+1}} \sim_\infty C_{x_{n+1},\alpha} + \log_{10}(m-1). \tag{25}$$

□

From Theorem 3.2, if the convergence zone is reached, then the digits common to two successive approximations x_n and x_{n+1} are also in common with the exact root α, up to $\log_{10}(m-1)$. If iterations are performed until the difference between two sucessive approximations x_n and x_{n+1} is not significant, then the significant digits of the last approximation x_{n+1} which are not affected by rounding errors are in common with the exact root α, up to $\delta = \lceil \log_{10}(m-1) \rceil$, as illustrated in Figure 1.

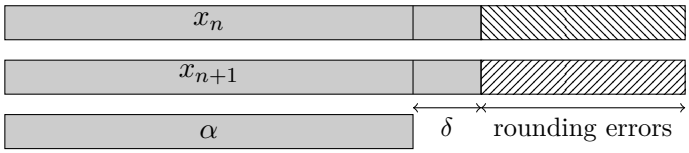

Figure 1: In case α is a multiple root, representation of the last iterates x_n and x_{n+1}, and the first digits of the exact root α: if the difference $x_n - x_{n+1}$ is not significant, then the digits of x_{n+1} which are not affected by rounding errors are in common with α, up to δ.

3.3 Numerical experiments

In the numerical experiments presented here, each polynomial root is approximated by computing a sequence (x_n) using Newton's method with DSA until the difference $x_n - x_{n+1}$ is not significant, *i.e.* $x_n \; ds= x_{n+1}$. The algorithm used is given below. We recall that in the SAM library all arithmetic and relational operators are redefined for stochastic variables. Therefore, as usual in C implementations of Newton's method, the stopping criterion is actually written as **while** (x!=y).

Algorithm 1: Newton using DSA

Data: x_{init} (initial approximation)
Result: x (approximation of a root of a polynomial f)
$x = x_{init}$;
do
  $y = x$;
  $x = y - f(y)/f'(y)$;
while $x \; ds\neq y$;

Let us first consider the following polynomial

$$P(x) = (19x+5)^5 (19x+21)^9 (19x+46)^{13} (19x+67)^{25}. \tag{26}$$

Its four roots with their multiplicities are presented in Table 1.

From Theorem 3.2 and Table 1, in the approximations of the roots obtained, the digits which are not affected by rounding errors are in common with the exact roots, up to 1 for α_1 and α_2, up to 2 for α_3 and α_4. This is observed in the numerical expriment described here, despite a few exceptions due to the probabilistic aspect of DSA.

roots	$\alpha_1 = -5/19$	$\alpha_2 = -21/19$	$\alpha_3 = -46/19$	$\alpha_4 = -67/19$
multiplicity m	5	9	13	25
$\lceil \log_{10}(m-1) \rceil$	1	1	2	2

Table 1: Roots of $P(x) = (19x+5)^5 (19x+21)^9 (19x+46)^{13} (19x+67)^{25}$ with their multiplities.

Figure 2 and Table 2 present the accuracy of the roots computed with the SAM library with respect to the precision used. For each root, 15 different precisions are considered, from 24 to 10,000 bits (equivalent to 3,010 decimal digits). In Figure 2 and Table 2 are reported: the accuracy estimated by SAM, *i.e.* the number of significant digits which are not affected by rounding errors, and the number of significant digits in common with the exact roots. As a remark, a log scale is used in Figure 2 for the y-axis to improve the readability of the results obtained with the lowest precisions. The accuracy is linear with respect to the decimal precision used and, as expected, it decreases as the multiplicity increases.

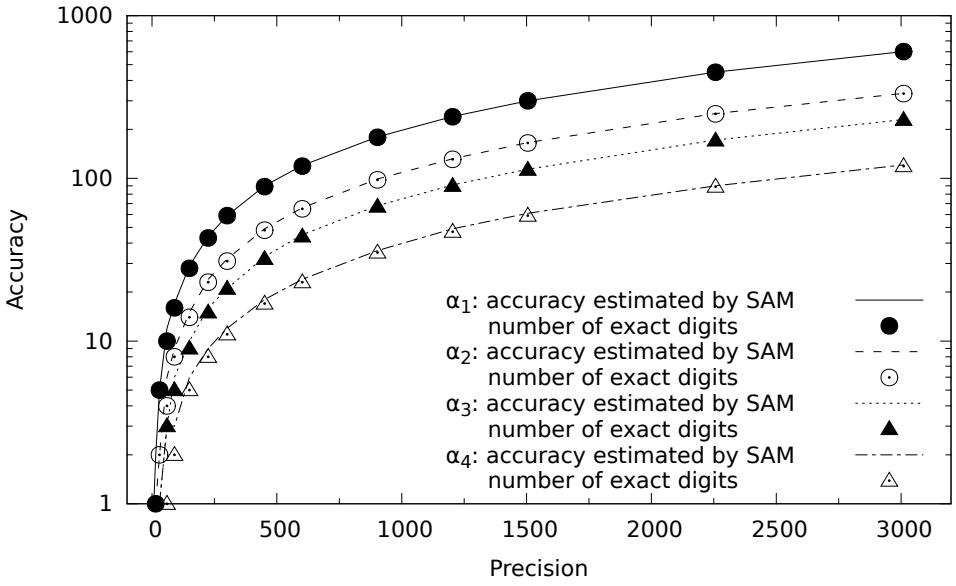

Figure 2: Accuracy of the roots of $P(x) = (19x+5)^5 (19x+21)^9 (19x+46)^{13} (19x+67)^{25}$ computed by SAM with respect to the decimal precision used.

The digits not affected by rounding errors which are provided by SAM are in common with the exact roots, up to 1 or 2. Actually the difference between the accuracy provided by SAM and the number of exact significant digits is 2 in six results out of sixty:

- one result is an approximation of α_1, its accuracy estimated by SAM is 12 digits and its exact accuracy, 10 digits, determined from Definition 3.1 is actually $\lfloor C_{x_{n+1},\alpha} \rfloor \approx \lfloor 10.83 \rfloor$ digits;
- one result is an approximation of α_2, its accuracy estimated by SAM is 6 digits and its exact accuracy, 4 digits, determined from Definition 3.1 is actually $\lfloor C_{x_{n+1},\alpha} \rfloor \approx \lfloor 4.92 \rfloor$ digits;
- in four results, which are approximations of α_4, this difference is consistent with Theorem 3.2 and Table 1.

A polynomial with a root of multiplicity 100 is also considered. In this case, from Theorem 3.2, in the approximations of the root obtained, the digits which are not affected by rounding errors are in common with

precision		$\alpha_1 = -5/19$		$\alpha_2 = -21/19$		$\alpha_3 = -46/19$		$\alpha_4 = -67/19$	
bits	digits	SAM	exact	SAM	exact	SAM	exact	SAM	exact
24	7	1	0	0	0	0	0	0	0
53	15	2	1	1	0	1	0	0	0
100	30	5	5	2	2	1	0	1	0
200	60	12	10	6	4	3	3	3	1
300	90	17	16	9	8	6	5	3	2
500	150	29	28	15	14	10	9	6	5
750	225	44	43	24	23	16	15	9	8
1000	301	59	59	32	31	21	21	12	11
1500	451	90	89	49	48	33	32	18	17
2000	602	120	119	66	65	45	44	24	23
3000	903	180	179	99	98	68	67	36	35
4000	1204	240	239	132	131	91	90	49	47
5000	1505	300	300	166	165	114	113	61	59
7500	2257	450	449	250	249	172	171	90	89
10000	3010	601	601	333	332	230	229	121	119

Table 2: For each root of $P(x) = (19x+5)^5\,(19x+21)^9\,(19x+46)^{13}\,(19x+67)^{25}$, number of decimal digits not affected by rounding errors estimated by SAM and number of decimal digits in common with the exact root.

the exact root, up to 2. Table 3 presents for $P(x) = (3x-1)^{100}$ the number of decimal digits not affected by rounding errors estimated by SAM and the number of decimal digits in common with the exact root $\alpha = 1/3$. The digits not affected by rounding errors which are provided by SAM are in common with the exact root up to 2, except in one case. Indeed, if the precision is set to 750 bits (equivalent to 225 decimal digits), the accuracy of the root estimated by SAM is 4 decimal digits and its exact accuracy, reported in Table 3, is 1 decimal digit. However this exact accuracy, determined from Definition 3.1 is actually $\lfloor C_{x_{n+1},\alpha} \rfloor \approx \lfloor 1.97 \rfloor$ digit.

Theorems 3.1 and 3.2 can be applied to other numerical examples presented in Section 4 that describes strategies to approximate polynomial roots along with their multiplicity.

4 Estimation of the multiplicity of polynomial roots

4.1 Previous work

Multiple roots of polynomials have been studied for a long time. See for example [18] and especially chapter 7 in [26].

In [2], Bini and Fiorentino used an adaptive multiprecision version of the Ehrlich-Aberth iteration [1] to compute clusters of roots. For that they use a suitable restarting criterion and cluster analysis to overcome the numerical instabilities arising from multiple roots. With their method, it is possible to count the number of roots in a cluster and so to have an approximation of the multiplicity.

More recently, Zeng [31, 32, 33] showed that it is possible to accurately compute a multiple root when one knows its multiplicity. This work is based on computing the pejorative manifold introduced by Kahan [16]. The algorithm does not require the use of multiprecision. It first computes the multiplicity structure by developing a numerical GCD-finder using a successive singular value updating and an iterative GCD refinement algorithm using a Gauss-Newton iteration.

precision		#digits		precision		#digits	
bits	digits	SAM	exact	bits	digits	SAM	exact
24	7	0	0	1500	451	6	4
53	15	1	0	2000	602	7	5
100	30	0	0	3000	903	10	8
200	60	1	0	4000	1204	13	11
500	150	3	1	5000	1505	17	14
750	225	4	1	7500	2257	24	22
1000	301	4	2	10000	3010	31	29

Table 3: For $P(x) = (3x-1)^{100}$, number of decimal digits not affected by rounding errors estimated by SAM and number of decimal digits in common with the exact root.

In [22], Rump proposed ten methods to bound multiple roots of polynomials. Given a univariate polynomial with a multiple root of multiplicity k, the paper discusses methods for computing a disc containing exactly k roots.

One can find reviews and presentations of the previous methods in the following books [20, 21]. In the sequel, we show how to compute polynomial roots with a requested accuracy using Newton's method and taking into account rounding errors. Then we present methods to compute the multiplicity of roots and therefore benefit from the quadratic convergence of the modified Newton's method.

4.2 Computation of polynomial roots with a requested accuracy

Algorithm 2 enables one to compute using DSA an approximation of a polynomial root, taking into account a requested accuracy, *i.e.* a minimal number of significant digits that will not be affected by rounding errors. Newton's method, implemented as in Algorithm 1, is applied iteratively until the desired root accuracy, computed from Equation 4, is reached. The precision, *i.e.* the number of digits used for the computation, is initially set to the requested accuracy multiplied by a rate arbitrarily chosen by the user, for instance 1.3. This rate enables one to take into account rounding errors that are necessarily generated; it avoids to perform two iterations in stable cases when one is sufficient. In accordance with the strategy described in [17], the precision is doubled at each iteration. If the requested accuracy and consequently the initial precision are too low, the root accuracy is also too low and useless iterations are performed. Therefore if the accuracy of the root computed at the first iteration is very low, a message is left to the user and the computation stops.

Algorithm 2: Newton with a requested accuracy using DSA

Data: $Requested\_accuracy$, x_{init} (initial approximation)
Result: x (root approximation with a requested accuracy)
$Precision = Requested\_accuracy * Rate$;
$step = 0$;
do
  $step = step + 1$;
  $x =$ Newton (x_{init});
  if $step = 1$ *and* $C_x \leq 2$ **then**
    print "increase requested accuracy" and stop ;
  end
  $x_{init} = x$;
  $Precision = Precision * 2$;
while $C_x \leq Requested\_accuracy$;

4.3 Estimation of the multiplicity from a linear regression

We now study the accuracy of the root compared to the precision of the computation. We will show that the relation is linear with a slope depending on the multiplicity.

Proposition 4.1 *Let f be a polynomial and let α be a root of f with multiplicity m. Let x be an approximation of α computed using Newton's method. Let $h = \alpha - x$ and $\varepsilon = \frac{f^{(m)}(\alpha)h^m}{m!}$. It follows that*

- *if $\alpha \neq 0$, then* $-\log_{10}|\varepsilon| = m\left(-\log_{10}\left|\frac{h}{\alpha}\right|\right) + \log_{10}\left|\frac{m!}{f^{(m)}(\alpha)\,\alpha^m}\right|$;
- *if $\alpha = 0$, then* $-\log_{10}|\varepsilon| = m\left(-\log_{10}|h|\right) + \log_{10}\left|\frac{m!}{f^{(m)}(\alpha)}\right|$.

Proof: Because α is a root of f with multiplicity m, $f(\alpha) = 0$, $\forall i \in \{1, ..., m-1\}$ $f^{(i)}(\alpha) = 0$ and $f^{(m)}(\alpha) \neq 0$. If $\varepsilon = \frac{f^{(m)}(\alpha)h^m}{m!}$, then

$$h^m = \frac{\varepsilon\, m!}{f^{(m)}(\alpha)}, \tag{27}$$

and

$$m\log_{10}|h| = \log_{10}\left|\frac{\varepsilon\, m!}{f^{(m)}(\alpha)}\right|. \tag{28}$$

If $\alpha \neq 0$, we deduce

$$-m\log_{10}\left|\frac{h}{\alpha}\right| = -\log_{10}\left|\frac{\varepsilon\, m!}{f^{(m)}(\alpha)\,\alpha^m}\right|, \tag{29}$$

and

$$-\log_{10}|\varepsilon| = m\left(-\log_{10}\left|\frac{h}{\alpha}\right|\right) + \log_{10}\left|\frac{m!}{f^{(m)}(\alpha)\,\alpha^m}\right|. \tag{30}$$

If $\alpha = 0$, from Equation 28 we deduce

$$-\log_{10}|\varepsilon| = m\left(-\log_{10}|h|\right) + \log_{10}\left|\frac{m!}{f^{(m)}(\alpha)}\right|. \tag{31}$$

□

From Proposition 4.1, the multiplicity m satisfies

$$P = m\,C_x + K_m, \tag{32}$$

where

- $P = -\log_{10}|\varepsilon|$. From Taylor expansion of f, $f(x) = \varepsilon + O(h^{m+1})$. Therefore $\varepsilon \approx f(x)$, x being an approximation of the root α. In general floating-point computations are affected by a relative error, the magnitude of which being the precision. This is similar as a perturbation of the given function f so that ε can be seen as the precision of the computation. As a consequence, P is assumed to be very close to the precision (*i.e.* the number of decimal digits) used for the computation.
- If $\alpha \neq 0$, $C_x = -\log_{10}\left|\frac{x-\alpha}{\alpha}\right|$ and if $\alpha = 0$, $C_x = -\log_{10}|x|$.
  In both cases C_x is the number of exact significant digits in x.
- If $\alpha \neq 0$, $K_m = \log_{10}\left|\frac{m!}{f^{(m)}(\alpha)\,\alpha^m}\right|$ and if $\alpha = 0$, $K_m = \log_{10}\left|\frac{m!}{f^{(m)}(\alpha)}\right|$.
  Because of the $\log_{10}$ function, K_m is considered not preponderant compared to $m\,C_x$.

If Algorithm 2 is used, we compute approximations of a polynomial root at several precisions. For each precision P_i, we obtain an approximation x_i and its accuracy C_{x_i} computed from Equation 4. From Equation 32, the different couples (C_{x_i}, P_i) are close to a line of slope m. Therefore the multiplicity m can be estimated from a linear regression. To enable the estimation of m even if one iteration in Algorithm 2 is sufficient, the line is assumed to include the origin: in this case both the precision and the accuracy are zero.

4.4 Estimation of the multiplicity from successive approximations

Here we assume that $\forall n \in \boldsymbol{N}, x_n \neq \alpha$. If the exact solution α is known, then we can determine the multiplicity m by performing successive euclidean divisions of the polynomial f by $(x-\alpha)$.

To estimate the multiplicity m of a polynomial root, Yakoubsohn [30] suggested a strategy based on the following property.

Proposition 4.2 *Let (x_n) be the sequence of approximations computed using Newton's method of the root α of multiplicity m of a polynomial f. Then*

$$\lim_{i\to\infty} \frac{x_{i+2}-x_{i+1}}{x_{i+1}-x_i} = 1 - \frac{1}{m}.$$

Proof: The Taylor expansion of f at α leads to

$$f(x) = f^{(m)}(\alpha)\frac{(x-\alpha)^m}{m!} + \varphi(x-\alpha)(x-\alpha)^{m+1} \text{ with } \lim_{x\to 0}\varphi(x) = 0. \tag{33}$$

Then, it follows that

$$f'(x) = f^{(m)}(\alpha)\frac{(x-\alpha)^{m-1}}{(m-1)!} + [(m+1)\varphi(x-\alpha) + (x-\alpha)\varphi'(x-\alpha)](x-\alpha)^m, \tag{34}$$

and

$$f'(x) = f^{(m)}(\alpha)\frac{(x-\alpha)^{m-1}}{(m-1)!} + \psi(x-\alpha)(x-\alpha)^m \text{ with } \lim_{x\to 0}\psi(x) = 0. \tag{35}$$

Therefore, from Equation 33, we have

$$\lim_{x\to\alpha} f(x) = f^{(m)}(\alpha)\frac{(x-\alpha)^m}{m!}, \tag{36}$$

and, from Equation 35,

$$\lim_{x\to\alpha} f'(x) = f^{(m)}(\alpha)\frac{(x-\alpha)^{m-1}}{(m-1)!}. \tag{37}$$

If $\alpha \neq 0$, then $\lim_{x\to\alpha} f'(x) \neq 0$. By division, we obtain

$$\frac{f(x)}{f'(x)} \approx_\alpha \frac{x-\alpha}{m}, \tag{38}$$

$$x - \alpha - \frac{f(x)}{f'(x)} \approx_\alpha x - \alpha - \frac{x-\alpha}{m}, \tag{39}$$

$$g(x) - \alpha \approx_\alpha \left(1 - \frac{1}{m}\right)(x-\alpha), \tag{40}$$

where $g(x) = x - f(x)/f'(x)$. Finally, it turns out that

$$g(x) \approx_\alpha \frac{\alpha}{m} + \left(1 - \frac{1}{m}\right)x. \tag{41}$$

Therefore if approximations x_i are close to the root α, the points of coordinates (x_i, x_{i+1}) are close to a line of slope $1 - \frac{1}{m}$ and $\lim_{i\to\infty} \frac{x_{i+2}-x_{i+1}}{x_{i+1}-x_i} = 1 - \frac{1}{m}$. $\square$

From Proposition 4.2, in the convergence phase, the multiplicity m can be estimated from three successive Newton's approximations.

4.5 Numerical experiments

In the numerical experiments presented here, several polynomial roots along with their multiplicities are computed using three strategies described below.

1. Newton's method is applied iteratively using Algorithm 2 until the accuracy of the approximated root is satisfactory. To estimate the multiplicity, a linear regression is performed using the different couples (accuracy, precision) obtained, as described in 4.3.

2. Algorithm 2 is used to compute the polynomial root with the requested accuracy. The root multiplicity is estimated from three successive approximations computed using Newton's method, as described in 4.4. For performance reasons, the multiplicity is estimated only at the first iteration of Algorithm 2.

3. The convergence of Newton's method is linear in case of a root multiplicity $m > 1$. However if the multiplicity is known, a quadratic convergence can be obtained using the modified Newton's method. In Algorithm 3, the root is approximated with a requested accuracy using firstly Newton's method and then the modified Newton's method. The multiplicity m is estimated from three successive approximations computed using Newton's method and then m is used in the modified Newton's method. Algorithm 4 shows how the modified Newton's method is implemented using DSA. Like in Algorithm 1, computations are performed until the difference between two successive iterates is a computational zero. Because of the fast convergence of the modified Newton's method, it is sometimes observed that, for an approximation x, $f(x)$ is a computational zero. In this case, further computations are useless. Such an approximation is returned as a solution.

Algorithm 3: Modified Newton with a requested accuracy using DSA

Data: $Requested\_accuracy$, x_{init} (initial approximation)
Result: x (root approximation with a requested accuracy)
$Precision = Requested\_accuracy * Rate$;
$step = 0$;
do
  $step = step + 1$;
  if $step = 1$ **then**
    $(x, m) = Newton(x_{init})$;
    if $C_x \leq 2$ **then**
      print "increase requested accuracy" and stop ;
    end
  else
    x = Modified_Newton(x_{init},m);
  end
  $x_{init} = x$;
  $Precision = Precision * 2$;
while $C_x \leq Requested\_accuracy$;

Algorithm 4: Modified Newton using DSA

Data: x_{init} (initial approximation), m (multiplicity)
Result: x (approximation of a root of a polynomial f)
$x = x_{init}$;
do
  if $f(x)\ ds= 0$ **then**
    return x and stop ;
  end
  $y = x$;
  $x = y - m * f(y)/f'(y)$;
while $x\ ds\neq y$;

The numerical experiments have been carried out using the SAM library on an Intel Core 2 Quad Q9550 processor at 2.83 GHz using the gcc version 4.9.2 compiler. In the numerical experiments presented here, the self-validation described in 2.2 is activated: all multiplications and divisions are controlled during the execution.

Tables 4 and 5 present results obtained with the polynomial $(3x - 1)^n$ for different values of the degree n that here equals the root multiplicity. In Algorithms 2 and 3, the initial approximation x_{init} and the rate used to initialize the precision are respectively set to 0.4 and 1.3.

In Table 4, for each degree n, the root is approximated using the three strategies previously described with the following requested accuracies: 10, 25, 50, 100, and 500 decimal digits. However no result is reported if the requested accuracy and consequently the initial precision are too low. In this case, a message is printed and the computation stops. If the degree n increases, the minimal requested accuracy also increases.

Table 4 presents the multiplicity m estimated using each strategy, the accuracy estimated by SAM, *i.e.* the number of significant digits which are not affected by rounding errors, the number of significant digits in common with the exact root, the execution time, and the number of steps performed in Algorithms 2 and 3. In the case of a simple root, it has been observed that, one step is sufficient; furthermore the modified Newton's method is useless. Therefore if $n = 1$, the results obtained using the third strategy are not reported: they are the same as those provided by the second one.

The first strategy and the second one provide the same root approximation with similar execution times. In accordance with Theorem 3.2, one can observe that, in this root approximation, the digits not affected

by rounding errors provided by SAM are always in common with the exact root, up to $\delta = \lceil \log_{10}(n-1) \rceil$ if $n > 1$. In the case of a simple root, *i.e.* when the polynomial considered is $3x - 1$, in accordance with Theorem 3.1, the accuracy estimated by SAM is the exact accuracy, except if 10 correct decimal digits are requested. However, in this case, the accuracy estimated by SAM is 12 digits and the exact accuracy, 11 digits, determined from Definition 3.1 is actually $\lfloor 11.95 \rfloor$ digits.

The first strategy and the second one differ by the estimation of the multiplicity. The estimation performed by the second strategy from three successive root approximations is always correct. It is better than the estimation performed by the first strategy that requires a linear regression. Furthermore with the second strategy, the multiplicity is correctly estimated at the first step of Algorithm 2, whereas the first strategy requires results computed at several steps.

Therefore with the third strategy, the multiplicity is estimated from three successive approximations computed by Newton's method and then the modified Newton's method can be used. One can observe that, with the third strategy, two steps of Algorithm 3 are sufficient; furthermore in the roots provided by SAM the digits not affected by rounding errors are always in common with the exact ones. Because of the quadratic convergence of the modified Newton's method, the third strategy performs better than the others: the execution times reported in Table 4 can be up to five orders of magnitude lower.

Table 5 presents results obtained with the polynomial $(3x - 1)^n$ for higher values of n. Because of the execution time, only the third strategy is used. Results computed with a sufficiently high requested accuracy are reported. One can observe that two steps of Algorithm 3 are sufficient, the multiplicity is correctly estimated and in the approximations provided by SAM the digits not affected by rounding errors are always in common with the exact root.

Results obtained using the third strategy with the polynomial P defined by Equation 26 are reported in Table 7. Table 6 presents for each root of the polynomial P, its multiplicity, the initial approximation x_{init} and the rate used to initialize the precision in Algorithm 3. Indeed a particular rate is set for each root. If the rate and consequently the initial precision are too low, the computed root may be numerical noise, *i.e.* a result with no correct digit because of rounding errors. If the rate is too high, the execution time may be unsatisfactory, because the computation may be performed with a precision that would be too high for the requested accuracy. In the same way we need an initial approximation of the root, we also need an approximation of the rate that can depend on the degree of the polynomial and on an approximation of the multiplicity. For each root approximation, the multiplicity is correctly determined. One can observe in Table 7 that, in the approximations provided by SAM, the digits not affected by rounding errors are always in common with the exact root, up to one. As a remark, all the approximations are obtained after two steps of Algorithm 3.

5 Conclusion and perspectives

In this article we show how to numerically compute polynomial roots along with their multiplicity taking into account a requested accuracy. We present a strategy to perform the optimal number of iterations with Newton's method. This strategy requires to control rounding error propagation using for instance Discrete Stochastic Arithmetic. Furthermore we show how to estimate in the root approximation obtained which digits are in common with the exact root. Two methods are compared to estimate the multiplicity of polynomial roots: one based on a linear regression from results computed with several precisions and one proposed by Yakoubsohn that uses three successive iterates of Newton's method. Yakoubsohn's method is particularly satisfactory and enables one to compute the root multiplicity and then benefit from the quadratic convergence of the modified Newton's method.

Attention should be paid to the initial precision to avoid both insignificant results and costly computations due to a too high precision. From the numerical experiments presented in this article, the initial precision should depend on the requested accuracy, the degree of the polynomial and a rough approximation of the multiplicity of the root. A perspective to this work consists in determining automatically the optimal initial precision. Another prospect would be to extend the theoretical results concerning the dynamical

n	requested accuracy	method	m	#digits SAM	#digits exact	time (s)	#steps
1	10	lin. regr.	1.08	12	11	5. E−3	1
1	10	succ. app.	1.00				
1	25	lin. regr.	1.04	31	31	6. E−3	1
1	25	succ. app.	1.00				
1	50	lin. regr.	1.01	64	64	6. E−3	1
1	50	succ. app.	1.00				
1	100	lin. regr.	1.01	129	129	7. E−3	1
1	100	succ. app.	1.00				
1	500	lin. regr.	1.00	649	649	8. E−3	1
1	500	succ. app.	1.00				
10	25	lin. regr.	9.91	26	25	1.1E−1	4
10	25	succ. app.	10.00				
10	25	modif.	10.00	35	35	1.1E−2	2
10	50	lin. regr.	9.96	52	51	4.5E−1	4
10	50	succ. app.	10.00				
10	50	modif.	10.00	70	70	2.4E−2	2
10	100	lin. regr.	9.98	104	103	2.2E+0	4
10	100	succ. app.	10.00				
10	100	modif.	10.00	142	142	5.6E−2	2
10	500	lin. regr.	10.00	520	519	1.5E+2	4
10	500	succ. app.	10.00				
10	500	modif.	10.00	714	714	1.3E+0	2
25	50	lin. regr.	24.66	83	82	3.4E+1	6
25	50	succ. app.	25.00				
25	50	modif.	25.00	66	66	3.5E−2	2
25	100	lin. regr.	25.32	166	165	2.3E+2	6
25	100	succ. app.	25.00				
25	100	modif.	25.00	133	133	1.0E−1	2
25	500	lin. regr.	25.00	833	831	1.8E+4	6
25	500	succ. app.	25.00				
25	500	modif.	25.00	675	675	2.9E+0	2
50	100	lin. regr.	49.43	167	165	3.8E+3	7
50	100	succ. app.	50.00				
50	100	modif.	50.00	131	131	1.5E−1	2
50	500	lin. regr.	49.99	833	831	2.1E+5	7
50	500	succ. app.	50.00				
50	500	modif.	50.00	662	662	5.5E+0	2

Table 4: Approximation of the root of $(3x-1)^n$ and estimation m of its multiplicity with several strategies: Newton's method and linear regression, Newton's method and estimation of the multiplicity from three successive approximations, modified Newton's method. Are reported: the number of decimal digits not affected by rounding errors estimated by SAM, the number of decimal digits in common with the exact root, the execution time, and the number of steps in Algorithms 2 and 3.

n	requested accuracy	m	#digits SAM	#digits exact	time (s)	#steps
100	100	100.00	130	130	1.1E−1	2
100	500	100.00	655	655	9.9E+0	2
100	1000	100.00	1311	1311	6.6E+1	2
200	500	200.00	653	653	1.6E+1	2
200	1000	200.00	1305	1305	1.2E+2	2
500	500	500.00	651	651	1.3E+1	2
500	1000	500.00	1301	1301	2.7E+2	2

Table 5: For $P(x) = (3x-1)^n$, estimation m of the root multiplicity and then computation of the root using modified Newton's method. Are reported: the number of decimal digits not affected by rounding errors estimated by SAM, the number of decimal digits in common with the exact root, the execution time, and the number of steps in Algorithm 3.

roots	$\alpha_1 = -5/19$	$\alpha_2 = -21/19$	$\alpha_3 = -46/19$	$\alpha_4 = -67/19$
multiplicity m	5	9	13	25
x_{init}	0	-1	-2	-3
$Rate$	3	5	7	12

Table 6: Roots of $P(x) = (19x+5)^5 (19x+21)^9 (19x+46)^{13} (19x+67)^{25}$ with their multiplities, values x_{init} (initial approximations) and $Rate$ used in Algorithm 3.

root	requested accuracy	#digits SAM	#digits exact	time (s)
α_1	50	62	61	3.0E−1
α_1	100	123	123	1.2E+0
α_1	200	243	243	5.1E+0
α_1	500	603	603	5.1E+1
α_2	50	69	68	7.9E−1
α_2	100	124	123	3.4E+0
α_2	200	235	235	1.9E+1
α_2	500	569	569	1.9E+2
α_3	50	80	79	1.7E+0
α_3	100	134	133	8.2E+0
α_3	200	242	241	4.5E+1
α_3	500	564	563	4.8E+2
α_4	50	73	72	5.2E+0
α_4	100	122	122	3.2E+1
α_4	200	218	218	2.0E+2
α_4	500	507	507	3.2E+3

Table 7: For each root of $P(x) = (19x+5)^5 (19x+21)^9 (19x+46)^{13} (19x+67)^{25}$, number of decimal digits not affected by rounding errors estimated by SAM, number of decimal digits in common with the exact root, and execution time.

control of Newton's method to the modified Newton's method. The two stopping criteria in Algorithm 4 should be taken into account to estimate the number of correct digits in the solution obtained.

References

[1] O. Aberth. Iteration methods for finding all zeros of a polynomial simultaneously. *Mathematics of Computation*, 27:339–344, 1973.

[2] D.A. Bini and G. Fiorentino. Design, analysis, and implementation of a multiprecision polynomial rootfinder. *Numerical Algorithms*, 23(2-3):127–173, 2000.

[3] J. Brajard, P. Li, F. Jézéquel, H.-S. Benavidès, and S. Thiria. Numerical Validation of Data Assimilation Codes Generated by the YAO Software. In *SIAM Annual Meeting, San Diego, California (USA)*, July 2013.

[4] J.-M. Chesneaux. *L'arithmétique stochastique et le logiciel CADNA*. Habilitation à diriger des recherches, Université Pierre et Marie Curie, Paris, France, November 1995.

[5] J.-M. Chesneaux and J. Vignes. Sur la robustesse de la méthode CESTAC. *Comptes Rendus de l'Académie des Sciences - Series I - Mathematics*, 307:855–860, 1988.

[6] J.-P. Dedieu. *Points fixes, zéros et la méthode de Newton*, volume 54 of *Mathématiques & Applications (Berlin)*. Springer, Berlin, 2006.

[7] P. Eberhart, J. Brajard, P. Fortin, and F. Jézéquel. High performance numerical validation using stochastic arithmetic. *Reliable Computing*, 21:35–52, 2015.

[8] S. Graillat. Accurate simple zeros of polynomials in floating point arithmetic. *Computers & Mathematics with Applications*, 56(4):1114–1120, 2008.

[9] S. Graillat, F. Jézéquel, S. Wang, and Y. Zhu. Stochastic Arithmetic in Multiprecision. *Mathematics in Computer Science*, 5(4):359–375, 2011.

[10] F. Jézéquel and J.-M. Chesneaux. CADNA: a library for estimating round-off error propagation. *Computer Physics Communications*, 178(12):933–955, 2008.

[11] F. Jézéquel, J.-M. Chesneaux, and J.-L. Lamotte. A new version of the CADNA library for estimating round-off error propagation in Fortran programs. *Computer Physics Communications*, 181(11):1927–1928, 2010.

[12] F. Jézéquel, J.-L. Lamotte, and O. Chubach. Parallelization of Discrete Stochastic Arithmetic on multicore architectures. In *10th International Conference on Information Technology: New Generations (ITNG), Las Vegas, Nevada (USA)*, April 2013.

[13] F. Jézéquel, J.-L. Lamotte, and I. Said. Estimation of numerical reproducibility on CPU and GPU. In *8th Workshop on Computer Aspects of Numerical Algorithms (CANA), Federated Conference on Computer Science and Information Systems (FedCSIS)*, pages 687–692, September 2015.

[14] F. Jézéquel, F. Rico, J.-M. Chesneaux, and M. Charikhi. Reliable computation of a multiple integral involved in the neutron star theory. *Mathematics and Computers in Simulation*, 71(1):44–61, 2006.

[15] H. Jiang, S. Graillat, C. Hu, S. Li, X. Liao, L. Cheng, and F. Su. Accurate evaluation of the k-th derivative of a polynomial and its application. *Journal of Computational and Applied Mathematics*, 243:28–47, 2013.

[16] W. Kahan. Conserving confluence curbs ill-condition. Technical Report 6, Computer Science Department, University of California, Berkeley, August 1972.

[17] V. Kreinovich and S.M. Rump. Towards optimal use of multi-precision arithmetic: a remark. *Reliable Computing*, 12:365–369, 2006.

[18] N. V. Kyurkchiev. *Initial approximations and root finding methods*, volume 104 of *Mathematical Research*. Wiley-VCH Verlag Berlin GmbH, Berlin, 1998.

[19] J.-L. Lamotte, J.-M. Chesneaux, and F. Jézéquel. CADNA_C: A version of CADNA for use with C or C++ programs. *Computer Physics Communications*, 181(11):1925–1926, 2010.

[20] J.M. McNamee. *Numerical methods for roots of polynomials. Part I*, volume 14 of *Studies in Computational Mathematics*. Elsevier B. V., Amsterdam, 2007.

[21] J.M. McNamee and V.Y. Pan. *Numerical methods for roots of polynomials. Part II*, volume 16 of *Studies in Computational Mathematics*. Elsevier/Academic Press, Amsterdam, 2013.

[22] S.M. Rump. Ten methods to bound multiple roots of polynomials. *Journal of Computational and Applied Mathematics*, 156(2):403–432, 2003.

[23] S.M. Rump and S. Graillat. Verified error bounds for multiple roots of systems of nonlinear equations. *Numerical Algorithms*, 54(3):359–377, 2010.

[24] N.S. Scott, F. Jézéquel, C. Denis, and J.-M. Chesneaux. Numerical 'health check' for scientific codes: the CADNA approach. *Computer Physics Communications*, 176(8):507–521, April 2007.

[25] F. Tisseur. Newton's method in floating point arithmetic and iterative refinement of generalized eigenvalue problems. *SIAM Journal on Matrix Analysis and Applications*, 22(4):1038–1057, 2001.

[26] J. F. Traub. *Iterative methods for the solution of equations*. Prentice-Hall Series in Automatic Computation. Prentice-Hall, Inc., Englewood Cliffs, N.J., 1964.

[27] J. Vignes. Zéro mathématique et zéro informatique. *Comptes Rendus de l'Académie des Sciences - Series I - Mathematics*, 303:997–1000, 1986. also: La Vie des Sciences, 4 (1) 1-13, 1987.

[28] J. Vignes. A stochastic arithmetic for reliable scientific computation. *Mathematics and Computers in Simulation*, 35:233–261, 1993.

[29] J. Vignes. Discrete Stochastic Arithmetic for validating results of numerical software. *Numerical Algorithms*, 37(1–4):377–390, December 2004.

[30] J.-C. Yakoubsohn. Numerical elimination, Newton method and multiple roots. In F. Chyzak, editor, *Algorithms Seminar 2001-2002*, pages 49–54. INRIA, 2003.

[31] Z. Zeng. A method computing multiple roots of inexact polynomials. In *Proceedings of the 2003 International Symposium on Symbolic and Algebraic Computation*, pages 266–272 (electronic). ACM, New York, 2003.

[32] Z. Zeng. Algorithm 835: MultRoot—a Matlab package for computing polynomial roots and multiplicities. *ACM. Transactions on Mathematical Software*, 30(2):218–236, 2004.

[33] Z. Zeng. Computing multiple roots of inexact polynomials. *Mathematics of Computation*, 74(250):869–903 (electronic), 2005.