2021 23rd International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC) | 978-1-6654-0650-5/21/$31.00 ©2021 IEEE | DOI: 10.1109/SYNASC54541.2021.00020

Computing multiple roots of polynomials in stochastic arithmetic with Newton method and approximate GCD

Stef Graillat*, Fabienne Jézéquel*†, Enzo Queiros Martins, and Maxime Spyropoulos

* Sorbonne Université, CNRS, LIP6, F-75005 Paris, France
Email: {Stef.Graillat, Fabienne.Jezequel}@lip6.fr
†Université Panthéon-Assas, 12 place du Panthéon, 75231 Paris CEDEX 05, France

Abstract—In this article, we propose new methods to compute multiple roots of polynomials in floating-point arithmetic. We rely on stochastic arithmetic that makes it possible to deal with rounding errors. We develop the concept of stochastic GCD that we use to deflate a polynomial in order to obtain a polynomial with single roots. We can then apply Newton method to get fast and accurate approximations of the roots. Numerical experiments show the effectiveness and efficiency of our methods.

Index Terms—approximate GCD, Discrete Stochastic Arithmetic, floating-point arithmetic, Newton method, numerical validation, rounding errors

1 Introduction

Polynomials appear in almost all areas of scientific computing. Generally the problem involved is to find roots of univariate polynomials. The wide range of use of polynomials needs to have fast and reliable methods to solve them. Roughly speaking, there are two general approaches: symbolic and numeric. The symbolic approach is based either on the theory of Gröbner basis or on the theory of resultants. For the numeric approach, we use iterative methods like Newton method or homotopy continuation methods. The symbolic algorithms give an exact representation of the result but can sometimes be slow. The numerical algorithms (performed in finite precision) are in general faster but give an approximate result. Our aim is to use numerical methods to get fast results without sacrificing accuracy.

If formulas are well-known for polynomials of degree less than 5, iterative methods are needed for greater degrees. In particular, Newton method is very used for its quadratic convergence. Other methods exist like Ehrlich-Aberth method, Durand-Kerner method or Laguerre method [1], [2]. In this article, we will only focus on Newton method.

When working with floating-point arithmetic, one major problem is to deal with ill-posed problems. Computing roots with multiplicities is an example of such a problem. Indeed, a small perturbation in the coefficients of a polynomial can change a double root into two single roots like for $P_\varepsilon(x) = (x-1)^2 - \varepsilon$. If $\varepsilon = 0$ then P_ε has 1 as a double root whereas for $\varepsilon > 0$, P_ε has two single roots $1 \pm \sqrt{\varepsilon}$. A similar phenomenon appears in the computation of the GCD of two polynomials. If $P_\varepsilon(x) = (x-1)^2 - \varepsilon$ and $Q(x) = (x-1)$, then $\gcd(P_\varepsilon(x), Q(x)) = x - 1$ for $\varepsilon = 0$ but $\gcd(P_\varepsilon(x), Q(x)) = 1$ for $\varepsilon \neq 0$. This has led to the development of different notions of approximate GCD or quasi-GCD [3]–[14]. There are various definitions for approximate GCD but they closely follow the same concept. Given two polynomials $P(x)$ and $Q(x)$ the approximate GCD is somehow related to the computation of the exact GCD of polynomials $\tilde{P}(x)$ and $\tilde{Q}(x)$ close to $P(x)$ and $Q(x)$ with the objective that this GCD has the greatest possible degree.

Computing such approximate GCD is generally expensive in terms of execution time. In order to avoid this, we propose in this article to compute a *stochastic GCD* (denoted as *st-gcd*) that makes it possible to smooth the computation of an approximate GCD using the classic Euclidean algorithm. It can be viewed as a kind of regularization technique that replaces floating-point arithmetic by stochastic arithmetic [15].

This stochastic GCD will be used for deflation in Newton method. Given a polynomial $P(x)$ with possible multiple roots, we will compute $G(x) = \text{st-gcd}(P(x), P'(x))$. We will then use $G(x)$ to compute the roots of $P(x)/G(x)$ which normally has only single roots.

The contributions of the paper can be sum up as follows.

- We use stochastic arithmetic to define a stochastic GCD that takes into account rounding errors when computations are performed in finite precision.
- We use the new GCD to deflate polynomials in order to obtain polynomials with single roots.
- We apply this to Newton's iterations and we show that we are able to accurately compute multiple roots of polynomials in finite precision.

The rest of the paper is organized as follows. In Section 2, we present how Discrete Stochastic Arithmetic (DSA) can be used to estimate rounding errors. Section 3 is devoted to the computation of multiple roots with DSA and the stochastic GCD. Numerical experiments are described in Section 4. Finally, concluding remarks are given in Section 5.

2 Estimation of rounding errors using Discrete Stochastic Arithmetic

This section presents the CESTAC method and Discrete Stochastic Arithmetic (DSA) as well as their implementations: CADNA and SAM. Those are well-known concepts and

978-1-6654-0650-5/21/$31.00 ©2021 IEEE
DOI 10.1109/SYNASC54541.2021.00020

we heavily rely on [16]. Further information can be found in [15] and [17].

2.1 Principles and validity of the CESTAC method

The CESTAC method [15] enables one to estimate the rounding error propagation which occurs with floating-point arithmetic. This probabilistic method uses a random rounding mode: at each elementary operation, the result is rounded up (towards $+\infty$) or down (towards $-\infty$) with the probability of 50 %. Hence, with this random rounding mode, the same program run several times provides different results. Therefore, the computer's deterministic arithmetic is replaced by a stochastic arithmetic, where each arithmetic operation is performed N times with the random rounding mode before the next one is executed. The CESTAC method supplies us with N samples $R_1, \ldots, R_N$ of the computed result R. The value of the computed result $\overline{R}$ is then chosen as the mean value of $\{R_i\}$ and, if no overflow occurs, its number of correct digits (*i.e.* its number of digits not affected by rounding errors) can be estimated as

$$C_R = \log_{10}\left(\frac{\sqrt{N}\,|\overline{R}|}{\sigma\tau_\beta}\right) \tag{1}$$

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

τ_β is the value of Student's distribution for $N-1$ degrees of freedom and a confidence level $1-\beta$. In practice, $\beta = 5\%$ and $N = 3$. Therefore the number of correct digits is estimated within a $1-\beta = 95\%$ confidence interval. It has been shown [15] that $N = 3$ is in some reasonable sense the optimal sample size. The estimation with $N = 3$ is more reliable than with $N = 2$ and increasing N does not significantly improve the quality of the estimation.

If both operands in a multiplication or the divisor in a division have no correct digits, the validity of C_R is compromised [15]. Therefore, the CESTAC method requires, during the execution of the user code, a dynamical control of multiplications and divisions, which is a so-called *self-validation* of the method.

2.2 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) [17] 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 Eq. 1. The self-validation of the CESTAC method also leads to the concept of computational zero [18] 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.

2.3 Accuracy estimation by CADNA and SAM

The CADNA[1] software [19], [20] 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 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.

The SAM library[2] [21] 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 [22]. 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. Such instabilities are usually generated by numerical noise, *i.e.* a result having no correct digits.

1. http://cadna.lip6.fr
2. http://www-pequan.lip6.fr/~jezequel/SAM

3 Computation of multiple roots of polynomials using DSA

3.1 Newton method in stochastic arithmetic

In section 3.2, an algorithm to compute multiple roots of polynomials in stochastic arithmetic will be described. This algorithm uses Newton method whose behaviour in stochastic arithmetic has been studied in [16]. We recall here properties of Newton method in stochastic arithmetic.

To compute the root α of a polynomial P, Newton method consists, from an initial approximation x_0, in computing the following sequence for $n \geq 0$:

$$x_{n+1} = x_n - \frac{P(x_n)}{P'(x_n)}. \quad (2)$$

In classical floating-point arithmetic, Newton's iterations are stopped thanks to a criterion such as $|x_{n+1} - x_n| \leq \varepsilon$, but it may be difficult to choose a suitable value for the parameter ε taking into account rounding errors. In stochastic arithmetic, the stopping criterion becomes $x_{n+1}\ ds{=}\ x_n$ in accordance with Definition 2.2. It means iterations are stopped when the difference between two successive iterates is a computational zero. Thus useless iterations that would bring numerical noise to the solution are avoided. Algorithm 1 presents a stochastic version of Newton method that relies on this optimal stopping criterion.

Algorithm 1: stochastic Newton method: st-Newton

Data: a polynomial P and an initial approximation x_0 of one of its roots
Result: an approximation x of a root of P

```
x = x_0;
do
    y = x;
    x = y − P(y)/P'(y);
while x ds≠ y;
```

The link between the exact root of a polynomial and the approximation provided by Algorithm 1 is discussed below. First, we need to make 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 $\mathbb{R}$ by

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

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.

The following theorem, introduced in [16], gives a relation between the common significant digits of two successive approximations of the root computed using Newton method and the common significant digits of an approximation and the root itself, when the root is *single*.

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

Then

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

According to Theorem 3.1, in the convergence zone, the digits that are common to two successive approximations are also in common with the exact root. In stochastic arithmetic, Newton's iterations are stopped when $x_{n+1}\ ds{=}\ x_n$, *i.e.* when $x_{n+1} - x_n$ is a computational zero (either numerical noise or the mathematical zero). In that case, the digits in x_{n+1} and x_n that are not affected by rounding errors are the same and these digits are in common with the exact root. Therefore, according to Theorem 3.1, in the approximation provided by Algorithm 1, the digits not affected by rounding errors that are estimated thanks to DSA are those of the exact root.

3.2 Algorithm based on GCD and Newton method

When used for the computation of a single root of a polynomial, the convergence of Newton method is quadratic. But it becomes linear for the computation of a multiple root. In that case, the convergence speed of modified Newton method is satisfactory, but it requires the root multiplicity.

We propose Algorithm 2 that takes benefit of both the convergence speed of Newton method and stochastic arithmetic to control rounding errors. To compute the (possibly multiple) roots of a polynomial P, first we compute G, the stochastic GCD of the polynomial P and its derivative P': $G = \text{st-gcd}(P, P')$ using Algorithm 4. Then we compute the polynomial $Q = P/G$ using a stochastic version of the Euclidean division (Algorithm 3). Algorithms 3 and 4 are stochastic versions of respectively the polynomial GCD and the polynomial Euclidean division. They use classic algorithms combined with the features of DSA, in particular the discrete stochastic relations recalled in Definition 2.2. The impact of DSA on these algorithms is described in Section 3.3.

The polynomial Q has normally only single roots and its degree d is its number of roots. Until a degree 4, its roots can be computed using adequate formulas, based on simple and well known expressions if d is 1 or 2, on Cardan's method if d is 3, and on Ferrari's method if d is 4 [23]. So, if $d \leq 4$, initial approximations of the roots are actually not required. From a degree 5, the stochastic version of Newton method (Algorithm 1) is used. The polynomial Q having single roots, Newton method exhibits a quadratic convergence.

An advantage of Algorithm 2 is the fact that the polynomial Q is computed once whatever the number of roots. Then all the roots are computed, either thanks to adequate formulas, or using Newton method applied for each root to the same low-degree polynomial.

3.3 Benefits of the DSA implementation

We exemplify the benefits of DSA with the computation of the root of a low-degree polynomial: $P(x) = (3x-1)^5$. The first step in Algorithm 2 is the computation of $G = \text{st-gcd}(P, P')$ using Algorithm 4. The polynomial R_2 in Algorithm 4 should be null, and the returned result should be $R_1 = P'$. Then, the polynomials G and Q in Algorithm 2 should be respectively of degree 4 and 1.

We compare two kinds of executions:

- with Algorithms 2, 3 and 4 using DSA and in particular stochastic relations recalled in Definition 2.2;

Algorithm 2: Computation of polynomial roots based on st-gcd and st-Newton

Data: a polynomial P and an array X_0 of initial approximations of its roots
Result: an array X of approximations of the roots

```
1  G = st-gcd(P, P');
2  Q = quotient(st-Euclidean-div(P, G));
3  if degree(Q) ≤ 4 then
4  |  computation of X using adequate formulas;
5  end
6  else
7  |  for i=1 to degree(Q) do
8  |  |  X[i]=st-Newton(Q, X_0[i]);
9  |  end
10 end
```

Algorithm 3: Stochastic polynomial Euclidean division: st-Euclidean-div

Data: polynomial A of degree n, polynomial B of degree m, with $0 \leq m \leq n$
Result: polynomials Q and R s.t. $A = B * Q + R$

```
1  R = A;
2  for i = n − m to 0 do
3  |  if lc(R) ds≠ 0 then
   |  |  // tests if degree(R) = m + i
   |  |  // lc(R): leading coefficient of R
4  |  |  q_i = lc(R)/lc(B);
5  |  |  R = R − q_i x^i B;
6  |  end
7  |  else
8  |  |  q_i = 0;
9  |  end
10 end
11 return Q = Σ_{i=0}^{n−m} q_i x^i and R
```

Algorithm 4: stochastic polynomial GCD: st-gcd

Data: polynomials A and B
Result: stochastic GCD of A and B

```
1  R_0 = A;
2  R_1 = B;
3  i = 1;
4  while R_i ds≠ 0 do
5  |  R_{i+1} =remainder (st-Euclidean-div(R_{i−1}, R_i));
   |  // R_{i+1} = R_{i−1} Mod R_i
6  |  i = i + 1;
7  end
8  return R_{i−1};
```

- with the same algorithms implemented using classic floating-point arithmetic. Because we focus on partial results obtained at the first steps of the computation, naive comparisons to zero are performed in Algorithms 3 and 4.

Table 1 presents results computed with/without DSA using a working precision of 35, 36 or 37 bits. We report the coefficients of the polynomial $R_2(x) = \sum_{i=0}^{3} R_2^i x^i$, the degree of the polynomials G and Q, and the computed root.

Without DSA, the computation is carried out using MPFR. Because we are particularly interested in the first iteration in Algorithm 4, the condition in line 4 is a naive comparison to zero. With 35 or 37 bits, all the coefficients of R_2 are zeroes, therefore the degrees of polynomials G and Q are correct. An approximation of the root is computed, but we have no information on its numerical quality. With 36 bits, three coefficients in R_2 are non-zeroes because of rounding errors. Undesirable iterations are performed in Algorithm 4 and consequently polynomials G and Q have incorrect degrees. It may be difficult to find a suitable stopping criterion in GCD computation that discards coefficients affected by rounding errors, but not small coefficients correctly computed. Furthermore the coefficients in R_i may be numerical noise with different orders of magnitude. For instance if the root of $(3x-1)^{10}$ is computed using Algorithm 2 with 36 bits, because of rounding errors, R_2 has non-zero coefficients with orders of magnitude that vary from 10^{-6} to 10^{-10}.

With the DSA implementation based on the SAM library, the condition in Algorithm 4 (on line 4) is satisfied if all the coefficients of the polynomial R_i are different from a computational zero in accordance with Definitions 2.1 and 2.2. This stopping criterion does not require any ε parameter and enables one to discard numerical noise whatever the working precision. Furthermore DSA estimates which digits in the results are not affected by rounding errors and displays only these correct digits. One can observe in Table 1 that the coefficients of the polynomial R_2 are identified as numerical noise, the degrees of G and Q are correctly computed, and the root is provided with 9 digits estimated as correct by DSA.

Algorithms 1 to 4 benefit from various features of DSA.

- In Algorithm 1, line 5, as already mentioned in Section 3.1, the criterion enables one to stop when the difference between two successive iterates is a computational zero, and consequently useless iterations are avoided.
- In Algorithm 2, if the degree d of the polynomial Q satisfies $d \leq 4$, its roots are directly computed using arithmetic expressions and the rounding errors affecting them are estimated by DSA. If $d > 5$, because Q has single roots, as already mentioned in Section 3.1, in each computed root, the digits estimated by DSA as not affected by rounding errors are those of the exact root.
- In Algorithm 3, line 3, the test is achieved by checking whether the leading coefficient of R is a computational zero.
- In Algorithm 4, line 4, polynomial coefficients that are computational zeroes are discarded.

	exact results	without DSA 35 bits	36 bits	37 bits	with DSA 35-37 bits
R_2^3	0	0	-1.86264514923e-9	0	@.0
R_2^2	0	0	9.31322574615e-10	0	@.0
R_2^1	0	0	-2.32830643654e-10	0	@.0
R_2^0	0	0	0	0	@.0
$degree(G)$	4	4	0	4	4
$degree(Q)$	1	1	5	1	1
root	1/3	3.33333333328e-1	Fail	3.333333333321e-1	0.333333333

TABLE 1
Computation of the root of $(3x-1)^5$ using Algorithm 2 with/without DSA

4 Numerical experiments

Numerical experiments have been carried to cover two aspects: the accuracy of the computed results and the performance. The platform for performance measurements is an Intel Core i7-8650U processor clocked at 1.9 GHz with 8 MB cache. The codes are compiled with gcc version 8.4.0 and optimized with the -O3 flag. The instability detection level chosen in the SAM library is the self-validation, already introduced in 2.1: all multiplications and divisions are controlled during the execution.

The roots of various polynomials have been computed using Algorithm 2 with the SAM library. The results have been compared with those also computed with SAM, but using an algorithm described in [16] and summarized below. In [16] a root of a polynomial P with multiplicity $m > 1$ is evaluated using modified Newton method that consists, from an initial approximation x_0, in computing the following sequence for $n \geq 0$:

$$x_{n+1} = x_n - m\frac{P(x_n)}{P'(x_n)}. \quad (3)$$

For multiple polynomial roots, modified Newton method exhibits a better convergence than Newton method. However it requires the evaluation of the multiplicity m. In [16] the multiplicity is determined thanks to three successive iterations of Newton method as proposed in [24]. Then the root is approximated using modified Newton method.

In Algorithm 2 the working precision depends on the requested accuracy and a rate both chosen by the user:

$$Precision = Requested\_accuracy * Rate \text{ with } Rate > 1. \quad (4)$$

Indeed because of rounding errors, the working precision has to be greater than the requested accuracy. In [16] the initial precision is set according to Eq. 4. Then the working precision is doubled between successive calls to modified Newton method applied iteratively.

Numerical experiments have been carried out with two types of polynomials.

- $P_n = (19x+5)^{n_1}(19x+21)^{n_2}(19x+46)^{n_3}(19x+67)^{n_4}$
  The roots of P_n are denoted as $\alpha_1 = -5/19$, $\alpha_2 = -21/19$, $\alpha_3 = -46/19$, and $\alpha_4 = -67/19$.
  The degrees of polynomials P_n have been chosen as follows.
  $n = \sum_{i=1}^{4} n_i = 54$ with $n_1 = 7$, $n_2 = 9$, $n_3 = 13$, $n_4 = 25$
  $n = \sum_{i=1}^{4} n_i = 104$ with $n_1 = 10$, $n_2 = 18$, $n_3 = 26$, $n_4 = 50$.
- $Q_n = (3x-2)^{n_1}(7x-3)^{n_2}(13x-4)^{n_3}(19x-2)^{n_4}(23x-1)^{n_5}$
  The roots of Q_n are denoted as $\beta_1 = 2/3$, $\beta_2 = 3/7$, $\beta_3 = 4/13$, $\beta_4 = 2/19$, and $\beta_5 = 1/23$.
  The degrees of polynomials Q_n have been chosen as follows.
  $n = \sum_{i=1}^{5} n_i = 55$ with $n_1 = 13$, $n_2 = 12$, $n_3 = 11$, $n_4 = 10$, $n_5 = 9$
  $n = \sum_{i=1}^{4} n_i = 105$ with $n_1 = 18$, $n_2 = 19$, $n_3 = 21$, $n_4 = 22$, $n_5 = 25$
  $n = \sum_{i=1}^{4} n_i = 5000$ with $n_i = 1000$ $(i = 1, \cdots, 5)$.

Table 2 presents results obtained using Algorithm 2, Tables 3 and 4 results computed using modified Newton method. As a recall, Algorithm 2 computes all the roots of a polynomial, whereas the algorithm based on modified Newton method requires an execution per root. Table 2 presents for polynomials P_{54}, P_{104}, Q_{55}, Q_{105}, and Q_{5000}:

- the requested number of decimal digits in all the roots,
- the number of digits in common with the exact roots; this number depending on the root, the minimum and the maximum number of digits are mentioned,
- the minimum rate that enables the accuracy requirement to be fulfilled for all the roots; this rate is determined starting from 1.1 and increasing by steps of 0.1,
- the execution time in seconds,
- the performance ratio w.r.t. the algorithm from [16] based on modified Newton method.

Poly.	#Digits requested	#Digits exact	Rate	Performance Time (s)	Performance Ratio
P_{54}	100	103-104	1.2	2.54e-03	7.9e+01
	500	533-534	1.1	4.54e-03	3.6e+03
	1000	1083-1084	1.1	9.88e-03	1.1e+04
P_{104}	100	109-110	1.3	3.78e-03	2.9e+02
	500	529-530	1.1	7.15e-03	1.7e+04
	1000	1079-1080	1.1	1.56e-02	N.A.
Q_{55}	100	109-111	1.3	3.63E-3	5.8e+01
	500	530-532	1.1	8.51E-3	1.1e+03
	1000	1079-1081	1.1	1.60E-2	3.5e+03
Q_{105}	100	107-109	1.3	5.04e-03	1.8e+02
	500	577-579	1.2	1.36e-02	5.6e+03
	1000	1276-1278	1.3	2.26e-02	2.3e+04
Q_{5000}	100	104-106	1.5	1.77e-01	N.A.
	500	503-506	1.1	3.35e-01	N.A.
	1000	1054-1056	1.1	6.40e-01	N.A.
	5000	5454-5456	1.1	5.00e+00	N.A.

TABLE 2
Computation of polynomial roots using Algorithm 2 with DSA

In Algorithm 2 after the execution of st-gcd and st-Euclidean-div, the four roots of polynomials P_{54} and P_{104} are computed using Ferrari's method [23], whereas the five roots of Q_{55}, Q_{105}, and Q_{5000} are computed using st-Newton. It can be observed in Table 2 that the accuracy requirement is fulfilled with a multiplicative rate between 1.1 and 1.5. For each root, the number of digits in common with the exact result is also the number of digits not affected by rounding errors estimated by DSA. This equality is in accordance with Theorem 3.1. Algorithm 2 outperforms the algorithm based on modified Newton method by 1 to 4 orders of magnitude. Concerning the roots of P_{104} with 1000 digits, the performance ratio is not available because the algorithm based on Newton method has failed to compute one of the roots. It must be pointed out that the first 1000 digits of the roots of polynomial Q_{5000} (of degree 5000) are computed in less than 1 second and the execution time becomes 5 seconds if 5000 digits are required.

Tables 3 and 4 present for each root of the polynomials P_{54}, P_{104}, Q_{55}, and Q_{105}:

- the requested number of decimal digits in the root,
- the number of digits not affected by rounding errors estimated by DSA,
- the number of digits in common with the exact root,
- the minimum rate that enables the accuracy requirement to be fulfilled; again this rate is determined starting from 1.1 and increasing by steps of 0.1,
- the execution time in seconds.

Poly.	Root	#Digits requested	#Digits DSA	#Digits exact	Rate	Time (s)
P_{54}	α_1	100	334	100	1.6	3.87e-02
	α_2		223	100	1.6	2.95e-02
	α_3		142	105	2.0	3.36e-02
	α_4		136	100	3.4	9.89e-02
	α_1	500	2018	525	1.8	7.20e-01
	α_2		2307	519	2.2	1.18e+00
	α_3		2900	512	3.0	2.51e+00
	α_4		5164	501	5.6	1.19e+01
	α_1	1000	4081	1037	1.8	3.84e+00
	α_2		4761	1005	2.2	6.18e+00
	α_3		6340	1005	3.1	1.50e+01
	α_4		11791	1015	6.0	8.15e+01
P_{104}	α_1	100	326	102	2.0	8.13e-02
	α_2		170	122	3.1	1.48e-01
	α_3		236	101	3.9	2.01e-01
	α_4		224	100	6.8	6.76e-01
	α_1	500	2530	502	2.4	2.83e+00
	α_2		3628	510	4.0	1.02e+01
	α_3		3673	501	5.0	1.64e+01
	α_4		7112	500	9.7	9.00e+01
	α_1	1000	5383	1023	2.5	1.67e+01
	α_2		8509	1018	4.3	6.60e+01
	α_3		10550	1007	5.8	1.43e+02
	α_4		Fail			

TABLE 3
Computation of polynomial roots using modified Newton method with DSA (polynomials with 4 roots)

It can be observed in Tables 3 and 4 that, for each root, the number of digits not affected by rounding errors estimated by DSA is greater than the number of digits in common with the exact result. The ratio between these numbers varies from 1.3 to 12. As a remark, Theorem 3.1 applies to the computation of roots of multiplicity 1 using Newton method,

Poly.	Root	#Digits requested	#Digits DSA	#Digits exact	Rate	Time (s)
Q_{55}	β_1	100	134	100	1.9	3.92e-02
	β_2		137	100	1.8	3.50e-02
	β_3		170	105	1.8	3.56e-02
	β_4		409	101	2.2	5.23e-02
	β_5		374	103	2.0	5.02e-02
	β_1	500	2957	503	3.0	2.52e+00
	β_2		2604	503	2.7	2.06e+00
	β_3		2612	512	2.6	1.88e+00
	β_4		2676	513	2.5	1.71e+00
	β_5		2364	506	2.2	1.22e+00
	β_1	1000	6359	1002	3.1	1.52e+01
	β_2		5990	1015	2.9	1.28e+01
	β_3		5662	1023	2.7	1.08e+01
	β_4		5871	1092	2.7	1.08e+01
	β_5		5026	1041	2.3	7.08e+00
Q_{105}	β_1	100	190	100	2.7	1.01e-01
	β_2		213	100	2.9	1.20e-01
	β_3		611	100	4.1	2.48e-01
	β_4		136	102	3.1	1.64e-01
	β_5		395	102	4.1	2.63e-01
	β_1	500	3260	506	3.8	8.33e+00
	β_2		3186	507	3.9	9.04e+00
	β_3		5089	508	5.1	1.75e+01
	β_4		4657	506	5.0	1.70e+01
	β_5		5570	501	5.8	2.48e+01
	β_1	1000	8081	1021	4.2	6.59e+01
	β_2		8131	1002	4.3	6.60e+01
	β_3		10642	1013	5.2	1.12e+02
	β_4		10493	1017	5.3	1.16e+02
	β_5		12251	1011	6.1	1.66e+02

TABLE 4
Computation of polynomial roots using modified Newton method with DSA (polynomials with 5 roots)

not to the computation of multiples roots using modified Newton method. The minimum multiplicative rate for the accuracy requirement to be fulfilled varies from 1.6 to 9.7 depending on the requested accuracy and the root multiplicity. In particular, a rate of 9.7 is required to obtain the first 500 digits of the root α_4 of P_{104} that has a multiplicity of 50. Modified Newton method fails to compute the same root with 1000 digits. With a rate of 10.9, the method provides the first 984 digits of α_4. But the working precision associated to a rate of 11, that is initially 11,000 digits and then increases, results in a failure due to the memory limit.

For all root approximations based on modified Newton method, the root multiplicity is correctly determined. However, then modified Newton method is used with a possibly high-degree polynomial (from degree 54 to 105 in our experiments). With Algorithm 2, Newton method is used with a low-degree polynomial (a polynomial of degree 4 or 5 for the results presented in Table 2).

5 Conclusion and perspectives

In this paper, we have presented an algorithm that efficiently and accurately computes polynomial roots, in particular multiple roots. This algorithm uses stochastic arithmetic that enables one to estimate rounding errors, and so to discard results that are actually numerical noise. With stochastic arithmetic, iterative algorithms can be stopped in an optimal way that does not rely on any parameter. Thanks to a stochastic version of the polynomial GCD algorithm and the polynomial Euclidean division, the proposed algorithm provides a low-degree polynomial with single roots. Depending

on the number of roots, they can be computed using either adequate formulas or Newton method. In the computed roots, stochastic arithmetic estimates which digits are not affected by rounding errors, and these digits are in common with the exact roots. W.r.t. an algorithm presented in [16] and based on modified Newton method, the proposed algorithm requires less working precision and less execution time: in our experiments the performance gain is up to 4 orders of magnitude.

The proposed algorithm takes into account an accuracy requirement and a multiplicative rate to determine the required working precision. From our experiments, this rate remains low and varies from 1.1 to 1.5. However it would be interesting to determine it automatically from information such as the required accuracy, the polynomial degree, and the multiplicity of the roots. Another perspective would be, for polynomials having at least 5 roots, the computation in parallel of their roots. Indeed, in this case, the roots are computed thanks to independent executions of Newton method.

Acknowledgment

This work was supported by the InterFLOP (ANR-20-CE46-0009) project of the French National Agency for Research (ANR).

References

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

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

[3] A. Schönhage, "Quasi-gcd computations," *J. Complexity*, vol. 1, no. 1, pp. 118–137, 1985.

[4] M.-T. Noda and T. Sasaki, "Approximate GCD and its application to ill-conditioned algebraic equations," in *Proceedings of the International Symposium on Computational Mathematics (Matsuyama, 1990)*, vol. 38, 1991, pp. 335–351.

[5] M.-A. Ochi, M.-T. Noda, and T. Sasaki, "Approximate greatest common divisor of multivariate polynomials and its application to ill-conditioned systems of algebraic equations," *J. Inform. Process.*, vol. 14, no. 3, pp. 292–300, 1991.

[6] N. K. Karmarkar and Y. N. Lakshman, "Approximate polynomial greatest common divisors and nearest singular polynomials," in *Proceedings of the 1996 international symposium on symbolic and algebraic computation, ISSAC '96*, Y. N. Lakshman, Ed., Jul. 1996, pp. 35–39.

[7] I. Z. Emiris, A. Galligo, and H. Lombardi, "Certified approximate univariate GCDs," *J. Pure Appl. Algebra*, vol. 117/118, pp. 229–251, 1997.

[8] B. Beckermann and G. Labahn, "When are two numerical polynomials relatively prime?" *J. Symbolic Comput.*, vol. 26, no. 6, pp. 677–689, 1998.

[9] ——, "A fast and numerically stable Euclidean-like algorithm for detecting relatively prime numerical polynomials," *J. Symbolic Comput.*, vol. 26, no. 6, pp. 691–714, 1998.

[10] P. Chin, R. M. Corless, and G. F. Corliss, "Optimization strategies for the approximate GCD problem," in *Proceedings of the 1998 international symposium on symbolic and algebraic computation, ISSAC '98*, O. Gloor, Ed., Aug. 1998, pp. 228–235.

[11] N. K. Karmarkar and Y. N. Lakshman, "On approximate GCDs of univariate polynomials," *J. Symbolic Comput.*, vol. 26, no. 6, pp. 653–666, 1998, symbolic numeric algebra for polynomials.

[12] V. Y. Pan, "Computation of approximate polynomial GCDs and an extension," *Inform. and Comput.*, vol. 167, no. 2, pp. 71–85, 2001.

[13] Z. Zeng, "The numerical greatest common divisor of univariate polynomials," in *Randomization, relaxation, and complexity in polynomial equation solving*, ser. Contemp. Math. Amer. Math. Soc., Providence, RI, 2011, vol. 556, pp. 187–217. [Online]. Available: https://doi.org/10.1090/conm/556/11014

[14] K. Nagasaka, "Toward the best algorithm for approximate GCD of univariate polynomials," *J. Symbolic Comput.*, vol. 105, pp. 4–27, 2021. [Online]. Available: https://doi.org/10.1016/j.jsc.2019.08.004

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

[16] S. Graillat, F. Jézéquel, and M. S. Ibrahim, "Dynamical Control of Newton's Method for Multiple Roots of Polynomials," *Reliable Computing*, vol. 21, pp. 117–139, 2016. [Online]. Available: http://interval.louisiana.edu/reliable-computing-journal/volume-21/reliable-computing-21-pp-117-139.pdf

[17] J. Vignes, "Discrete Stochastic Arithmetic for validating results of numerical software," *Numerical Algorithms*, vol. 37, no. 1–4, pp. 377–390, Dec. 2004.

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

[19] F. Jézéquel and J.-M. Chesneaux, "CADNA: a library for estimating round-off error propagation," *Computer Physics Communications*, vol. 178, no. 12, pp. 933–955, 2008.

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

[21] S. Graillat, F. Jézéquel, S. Wang, and Y. Zhu, "Stochastic Arithmetic in Multiprecision," *Mathematics in Computer Science*, vol. 5, no. 4, pp. 359–375, 2011. [Online]. Available: http://dx.doi.org/10.1007/s11786-011-0103-4

[22] L. Fousse, G. Hanrot, V. Lefèvre, P. Pélissier, and P. Zimmermann, "MPFR: A multiple-precision binary floating-point library with correct rounding," *ACM Trans. Math. Softw.*, vol. 33, no. 2, pp. 13:1–13:15, 2007, http://www.mpfr.org.

[23] A. Kurosh, *Higher algebra*. "Mir", Moscow, 1988, translated from the Russian by George Yankovsky, Reprint of the 1972 translation.

[24] J.-C. Yakoubsohn, "Numerical elimination, Newton method and multiple roots," in *Algorithms Seminar 2001-2002*, F. Chyzak, Ed. INRIA, 2003, pp. 49–54. [Online]. Available: http://algo.inria.fr/seminars