# Global Newton Iteration over Archimedean and non-Archimedean Fields

**(Full Version)**

Jonathan D. Hauenstein[1]*, Victor Pan[2]**, and Agnes Szanto[1]***

[1] North Carolina State University,
{hauenstein,aszanto}@ncsu.edu
[2] Lehman College - City University of New York,
victor.pan@lehman.cuny.edu

**Abstract.** In this paper, we study iterative methods on the coefficients of the *rational univariate representation (RUR)* of a given algebraic set, called *global Newton iteration*. We compare two natural approaches to define locally quadratically convergent iterations: the first one involves Newton iteration applied to the approximate roots individually and then interpolation to find the RUR of these approximate roots; the second one considers the coefficients in the exact RUR as zeroes of a high dimensional map defined by polynomial reduction, and applies Newton iteration on this map. We prove that over fields with a p-adic valuation these two approaches give the same iteration function, but over fields equipped with the usual Archimedean absolute value, they are not equivalent. In the latter case, we give explicitly the iteration function for both approaches. Finally, we analyze the parallel complexity of the different versions of the global Newton iteration, compare them, and demonstrate that they can be efficiently computed. The motivation for this study comes from the certification of approximate roots of overdetermined and singular polynomial systems via the recovery of an exact RUR from approximate numerical data.

## 1 Introduction

Let $F_1, \ldots, F_n \in \mathbb{K}[x_1, \ldots, x_n]$ be polynomials with coefficients from a field $\mathbb{K}$, $\mathcal{J} := \langle F_1, \ldots, F_n \rangle$ the ideal they generate, and assume that $\mathcal{J}$ is zero dimensional and radical. We consider two cases for the coefficient field $\mathbb{K}$:

**Non-Archimedean case:** Let $R$ be a principal ideal domain, $\mathbb{K}$ its field of fractions, and $p$ an irreducible element in $R$. Then, we can equip $\mathbb{K}$ with the $p$-adic valuation, which defines a non-Archimedean metric on vector spaces over $\mathbb{K}$. The main examples include $R = \mathbb{Z}$ and $p$ a prime number, or $R =$

* Research was partly supported by NSF grant DMS-1262428 and DARPA Young Faculty Award.

** Research was partly supported by NSF grant CCF-1116736.

*** Research was partly supported by NSF grant CCF-1217557.

$\mathbb{Q}[t]$ and $p = t$. Note that after clearing denominators, we can assume that $F_1, \ldots, F_m \in R[x_1, \ldots, x_n]$.

**Archimedean case:** In this case, $\mathbb{K}$ is a subfield of $\mathbb{C}$ and it is equipped with the usual absolute value. The usual Euclidean norm defines an Archimedean metric on vector spaces over $\mathbb{K}$.

The objective of this paper is to study iterative methods on the coefficients of the *rational univariate representation (RUR)* of a component of $\mathcal{J}$, and compare them in the Archimedean and the non-Archimedean cases. The RUR of a component of $\mathcal{J}$, originally defined in [32], is a simple representation of a subset of the common roots of $F_1, \ldots, F_n$, expressing the coordinates of these common roots as Lagrange interpolants at nodes which are given as the roots of a univariate polynomial (see definition below).

We study two natural approaches for iterations that are locally quadratically convergent to an exact RUR of a component of $\mathcal{J}$, both based on Newton's method:

- To update an RUR, apply the usual $n \times n$ Newton iteration to each common root of the old RUR, and compute the updated RUR which defines these updated roots. In this approach we assume inductively that the common roots of the iterated RUR's are known exactly.
- Consider the map that takes an RUR and returns the reduced form of the input polynomials $F_1, \ldots, F_n$ modulo the RUR. Since an exact RUR of a component of $\mathcal{J}$ is a zero of this map, we apply Newton's method to this map.

Note that in the $p$-adic case, the first iteration was studied in [15] where they gave the iteration function explicitly and analyzed its complexity in terms of straight-line programs, while the second approach was proposed in [38], without giving the iteration explicitly.

The main results of this paper are as follows: First we prove that the above two approaches give the same iteration function in the p-adic valuation. Next, we show that in the Archimedean case the two iterations are not equivalent. In this case, we give the explicit iteration functions for both approaches and show that they are also different from the iteration function presented in [15] and interpreted for the Archimedean case. We illustrate the methods on an example involving the mobility of a spacial mechanism. Finally, we analyze the parallel complexity of both approaches in the Archimedean case: For the first approach, we use $n \times n$ Newton iterations independently for each root and an efficient parallel Vandermonde linear solver for Lagrange interpolation. For the second approach we utilize efficient parallel Toeplitz-like linear system solvers to compute modular inverses of univariate polynomials. We present a version of the algorithms of [30] to solve Toeplitz-like linear systems that uses a more efficient displacement representation with factor circulant matrices defined in [31, Example 4.4.2] rather than triangular Toeplitz matrices. Finally, we briefly discuss the computation of modular inverses and cofactors when all the roots of at least one of the two associated input polynomials are simple and known.

### 1.1 Related work

The motivation to study numerical approximations of RUR's come from a work in progress in [2] to certify approximate roots of overdetermined and singular polynomial systems over $\mathbb{Q}$. For well-constrained non-singular systems, Smale's $\alpha$-theory (see [6, Chapters 8 and 14]) gives a tool for the certification of approximate roots, as was explored and implemented in **alphaCertified** [18]. However, **alphaCertified** does not straightforwardly extend to overdetermined or singular systems: in [18], they propose to use universal lower bounds for the minimum of positive polynomials on a disk, such as in [22], but they conclude that such bounds are "too small to be practical." To overcome this difficulty, in [2] it is proposed to iteratively compute the exact RUR of a rational component from approximations of the roots, and then use the machinery of [18] to certify approximate roots of this RUR. While [2] is devoted to considerations about the global behavior of the iteration, this paper considers different choices of the iteration function and their parallel complexity.

The iterative algorithms that are in the core of this paper are the Archimedean adaptations of what is known as "global Newton iteration" or "multivariate Hensel lifting" or "Newton-Hensel lifting" in the computer algebra literature, where it is defined for the non-Archimedean case. Various versions of the Newton-Hensel lifting were applied in many applications within computer algebra, including in univariate polynomial factorization [42, 27], multivariate polynomial factorization [8, 16, 23], gcd of sparse multivariate polynomials [24], lexicographic and general Gröbner basis computation of zero dimensional algebraic sets [38, 41], geometric resolution of equi-dimensional algebraic sets [13, 14, 19, 15], Chow forms [21], and sparse interpolation [3]. As we mentioned above, the most related to this paper are [38, 15].

Computing numerical approximation to symbolic objects in the Archimedean metric is not new either. There is a significant literature studying such hybrid symbolic-numeric algorithms, and without trying to give a complete bibliography, the following mentions the papers that are the closest to our work.

Closest to our approach is the literature on finding the vanishing ideal of a finite point set given with limited precision. In [7], they give an algorithm that given *one* approximate zero of a polynomial system, finds the RUR of the irreducible component containing the corresponding exact roots in randomized polynomial time. The algorithm in [7] applies the univariate results of [25] using lattice basis reduction. The main point of our approach in this paper and in [2] is that we assume to know *all* approximate roots of a rational component, so in this case we can compute the exact RUR much more efficiently, and in parallel.

The papers [7, 35, 29, 1, 20, 9, 10] use a more general approach than the one here, by computing border bases for a given set of approximate roots, which avoids defining a random primitive element as is done for RURs used in this paper. For general polynomial systems, numerical computation of Gröbner bases was proposed, for example, in [33, 34, 28, 37]. The focus of these papers is to find numerically stable support for the bases, which we assume to be given here by the primitive element.

## 2 Preliminaries

Let us start with recalling the notion of Roullier's *Rational Univariate Representation (RUR)*, originally defined in [32]. Instead of defining the RUR of an ideal $\mathcal{J}$, here we only define the notion of the *RUR of a component of* $\mathcal{J}$, which is a weaker notion. We follow the notation in [2].

Let $\mathbb{K}$ be a field. Given $\mathbf{F} = (F_1, \dots, F_n) \subset \mathbb{K}[x_1, \dots, x_n]$ for some $n$, and assume that the ideal $\mathcal{J} := \langle F_1, \dots, F_n \rangle$ is radical and zero dimensional. Then the factor ring $\mathbb{K}[x_1, \dots, x_n]/\mathcal{J}$ is a finite dimensional vector space over $\mathbb{K}$, and we denote

$$\delta := \dim_{\mathbb{K}} \mathbb{K}[x_1, \dots, x_n]/\mathcal{J}.$$

Furthermore, for almost all $(\lambda_1, \dots, \lambda_n) \in \mathbb{K}^n$ (except a Zariski closed subset), the linear combination

$$u(x_1, \dots, x_n) := \lambda_1 x_1 + \cdots + \lambda_n x_n$$

is a *primitive element* of $\mathcal{J}$, i.e. the powers $1, u, u^2, \dots, u^{\delta-1}$ form a linear basis for $\mathbb{K}[x_1, \dots, x_n]/\mathcal{J}$ (c.f. [32]).

In the algorithms that follow, we compute an RUR that may not generate the ideal $\mathcal{J}$, nevertheless the polynomials $F_1, \dots, F_n$ vanish modulo the RUR. In this case the RUR will generate a *component of* $\mathcal{J}$. We have the following definition:

**Definition 1.** *Let $\mathcal{J} = \langle F_1, \dots, F_n \rangle \subset \mathbb{K}[x_1, \dots, x_n]$ be as above. Let $\lambda_1 x_1 + \cdots + \lambda_n x_n$ be a primitive element of $\mathcal{J}$. We call the polynomials*

$$T - \lambda_1 x_1 + \cdots + \lambda_n x_n,\ q(T),\ v_1(T), \dots, v_n(T) \tag{1}$$

*a* Rational Univariate Representation (RUR) of a component of $\mathcal{J}$ *if it satisfies the following properties:*

- *$q(T) \in \mathbb{K}[T]$ is a monic polynomial of degree $d \leq \delta$,*
- *$\gcd_T(q(T), q'(T)) = 1$ where $q'(T) = \frac{\partial q(T)}{\partial T}$,*
- *$v_1(T), \dots, v_n(T) \in \mathbb{K}[T]$ are all degree at most $d-1$ and satisfy*

$$\lambda_1 v_1(T) + \cdots + \lambda_n v_n(T) = T,$$

- *for all $i = 1, \dots, n$ we have*

$$F_i(v_1(T), \dots, v_n(T)) \equiv 0 \mod q(T).$$

First note that the set

$$\{q(T),\ x_1 - v_1(T), \dots, x_n - v_n(T)\} \tag{2}$$

forms a Gröbner basis for the ideal it generates with respect to the lexicographic monomial ordering with $T < x_1 < \cdots < x_n$.

Next let us recall the relationship between the RUR of a component of $\mathcal{J}$ and its (exact) roots. Let

$$V(\mathcal{J}) = \{\xi_1, \dots, \xi_\delta\} \subset \mathbb{C}^n$$

be the set of common roots of $\mathcal{J}$. Denote $\xi_i = (\xi_{i,1}, \dots, \xi_{i,n})$ for $i = 1, \dots, \delta$. Then for any $n$-tuple $(\lambda_1, \dots \lambda_n) \in \mathbb{K}^n$ such that for $i, j = 1, \dots, \delta$

$$\lambda_1 \xi_{i,1} + \dots + \lambda_n \xi_{i,n} \neq \lambda_1 \xi_{j,1} + \dots + \lambda_n \xi_{j,n} \quad \text{if } i \neq j,$$

we can define the primitive element $u = \lambda_1 x_1 + \dots + \lambda_n x_n$ for $\mathcal{J}$. Since all roots of $\mathcal{J}$ are distinct, such primitive element exist, and can be computed from the roots $\{\xi_1, \dots, \xi_\delta\}$, or using randomization. Fix such $(\lambda_1, \dots \lambda_n) \in \mathbb{K}^n$. For $d \leq \delta$ let $\{\xi_1, \dots, \xi_d\}$ be a subset of $V(\mathcal{J})$ and define

$$\mu_i := \lambda_1 \xi_{i,1} + \dots + \lambda_n \xi_{i,n}, \quad i = 1, \dots d. \tag{3}$$

The RUR of the component of $\mathcal{J}$ corresponding to the subset $\{\xi_1, \dots, \xi_d\} \subset V(\mathcal{J})$ is defined by

$$q(T) := \prod_{i=1}^{d} (T - \mu_i), \tag{4}$$

and for each $j = 1, \dots, n$, the polynomial $v_j(T)$ is the unique Lagrange interpolant of degree at most $d - 1$ satisfying

$$v_j(\mu_i) = \xi_{i,j} \quad \text{for } i = 1, \dots, d. \tag{5}$$

Note that if $\mathcal{J}$ is defined by polynomials over $\mathbb{K}$, then the polynomials in the RUR of $\mathcal{J}$ have coefficients in $\mathbb{K}$, but that is not true for all components of $V(\mathcal{J})$. We call a subset $\{\xi_1, \dots, \xi_d\} \subset V(\mathcal{J})$ a *rational component of* $\mathcal{J}$ if the corresponding RUR has also coefficients in $\mathbb{K}$.

## 3 Global Newton Iteration

In this section we describe iterative methods that improves the accuracy of the coefficients of the RUR of a component of $\mathcal{J}$. We use a similar approach as in [15, Section 4], but instead of a coefficient ring with the p-adic absolute value, here we make adaptations to coefficient field $\mathbb{K} \subseteq \mathbb{C}$ equipped with the usual absolute value. We start with recalling the definitions given in [15, Section 4].

### 3.1 Non-Archimedean Global Newton iteration

First, we briefly describe the global Newton iteration defined in [15, Section 4]. There the coefficient domain is the ring $\mathbb{Q}[t]$ and the non-Archimedean metric is defined by the irreducible element $t \in \mathbb{Q}[t]$. They consider a square system $\boldsymbol{F} = (F_1, \ldots, F_n)$ with $F_i \in \mathbb{Q}[t][x_1, \ldots, x_n]$. Let

$$u(x_1, \ldots, x_n) = \lambda_1 x_1 + \cdots + \lambda_n x_n = T$$

be a random primitive element for $\langle F_1, \ldots, F_n \rangle$. Furthermore, define

$$I := \langle t^k \rangle \text{ for some } k.$$

In [15, Section 4] they assume that some initial approximate RUR is given for a component of $\mathcal{J}$ :

$$q(T),\ \mathbf{v}(T) := (v_1(T), \ldots, v_n(T)) \in \mathbb{Q}[t][T]$$

satisfying the following assumptions:

**Assumption 2** Let $\boldsymbol{F}$, $u$, $I$, $q(T)$ and $\mathbf{v}(T)$ be as above. Then

1. $q(T)$ is monic and has degree $d$,
2. $v_i(T)$ has degree at most $d-1$,
3. $\boldsymbol{F}(\mathbf{v}(T)) \equiv 0 \mod q(T) \mod I$,
4. $\lambda_1 v_1(T) + \cdots + \lambda_n v_n(T) = T \mod I$,
5. $J_{\boldsymbol{F}}(\mathbf{v}(T)) := \left[\frac{\partial F_i}{\partial x_j}(\mathbf{v}(T))\right]_{i,j=1}^{n}$ is invertible modulo $q(T)$ and $I$.

They define the following updates:

**Definition 3.** *Assume that $\boldsymbol{F}$, $u$, $q(T)$, $\mathbf{v}(T)$ and $I$ satisfy Assumption 2. Then in [15, Section 4] they define*

$$\begin{aligned}
u &= \sum_{i=1}^{n} \lambda_i x_i = T \textit{ remains the same as for the initial RUR,}\\
\boldsymbol{w}(T) &:= \mathbf{v}(T) - \left(J_{\boldsymbol{F}}(\mathbf{v}(T))^{-1} \boldsymbol{F}(\mathbf{v}(T)) \bmod q(T)\right) \bmod I^2,\\
\Delta(T) &:= \sum_{i=1}^{n} \lambda_i w_i(T) - T \mod I^2,\\
\boldsymbol{V}(T) &:= \boldsymbol{w}(T) - \left(\Delta(T) \cdot \frac{\partial \boldsymbol{w}(T)}{\partial T} \mod q(T)\right) \bmod I^2,\\
Q(T) &:= q(T) - \left(\Delta(T) \cdot \frac{\partial q(T)}{\partial T} \mod q(T)\right) \mod I^2.
\end{aligned}$$

In [15, Section 4] they prove the following:

**Proposition 4 ([15]).** *Assume that $\boldsymbol{F}$, $u$, $q(T)$, $\mathbf{v}(T)$ and $I$ satisfy Assumption 2 and let $\boldsymbol{w}(T)$, $\Delta(T)$, $\boldsymbol{V}(T)$, $Q(T)$ be as in Definition 3. Then*

*(i)* $\mathbf{v}(T) \equiv \boldsymbol{w}(T) \equiv \boldsymbol{V}(T)$, $q(T) \equiv Q(T)$ *and* $\Delta(T) \equiv 0 \mod I$
*(ii)* $\boldsymbol{F}(\boldsymbol{w}(T)) \equiv 0 \mod q(T) \mod I^2$,
*(iii)* $\langle q(T), U - T - \Delta(T), x_1 - w_1(T), \ldots, x_n - w_n(T)\rangle = \langle Q(U), T - U - \Delta(U), x_1 - V_1(U), \ldots, x_n - V_n(U)\rangle \mod I^2$,
*(iv)* $\boldsymbol{F}(\boldsymbol{V}(T)) \equiv 0 \mod Q(T) \mod I^2$,
*(v)* $\lambda_1 V_1(T) + \cdots + \lambda_n V_n(T) = T \mod I^2$.

### 3.2 First Construction

Our first variation of Definition 3 will have the property that it agrees to the approximate RUR obtained from the approximate roots via Lagrange interpolation as was described in the Preliminaries. We give our definition over some general coefficient ring $R$, but later we will use $R = \mathbb{K} \subset \mathbb{C}$, or $\mathbb{Q}[t]/I^2$. We need the following assumptions:

**Assumption 5** Let $\boldsymbol{F} = (F_1, \ldots, F_n)$, $u = \lambda_1 x_1 + \cdots + \lambda_n x_n$, $q(T)$ and $\mathbf{v}(T) = (v_1(T), \ldots, v_n(T))$ polynomials over some Euclidean domain $R$ as above. We assume that

1. $q(T)$ is monic and has degree $d$,
2. $v_i(T)$ has degree at most $d-1$,
3. $\frac{\partial q(T)}{\partial T}$ is invertible modulo $q(T)$,
4. $\lambda_1 v_1(T) + \cdots + \lambda_n v_n(T) = T$,
5. $J_{\boldsymbol{F}}(\mathbf{v}(T)) := \left[\frac{\partial F_i}{\partial x_j}(\mathbf{v}(T))\right]_{i,j=1}^{n}$ is invertible modulo $q(T)$.

Our first construction for the update is defined as follows:

**Definition 6.** *Assume that $\boldsymbol{F}$, $u$, $q(T)$ and $\mathbf{v}(T)$ satisfy Assumption 5. Then we define*

$$u = \sum_{i=1}^{n} \lambda_i x_i = T$$

$$\boldsymbol{w}(T) := \mathbf{v}(T) - \left(J_{\boldsymbol{F}}(\mathbf{v}(T))^{-1}\boldsymbol{F}(\mathbf{v}(T)) \mod q(T)\right),$$

$$\Delta(T) := \sum_{i=1}^{n} \lambda_i w_i(T) - T \textit{ so far the same as in Definition 3,} \tag{6}$$

$$\tilde{\boldsymbol{V}}(T + \Delta(T)) = \begin{bmatrix} \tilde{V}_1(T+\Delta(T)) \\ \vdots \\ \tilde{V}_n(T+\Delta(T)) \end{bmatrix} := \boldsymbol{w}(T) \mod q(T), \tag{7}$$

$$\Delta\tilde{Q}(T+\Delta(T)) := -(T+\Delta(T))^d \mod q(T) \tag{8}$$

$$\tilde{Q}(T) := \Delta\tilde{Q}(T) + T^d \tag{9}$$

Note that in Definition 6 we define $\tilde{\boldsymbol{V}}(T+\Delta(T))$ and not $\tilde{\boldsymbol{V}}(T)$, but the coefficients of $\tilde{V}_i(T)$ can be obtained as solutions of linear systems. Similarly for the coefficients of $\Delta\tilde{Q}(T)$. In the next proposition we examine these linear systems and give conditions on the existence and uniqueness of their solutions.

**Proposition 7.** *The coefficients of the polynomials $\tilde{V}_i(T)$ in (7) for $i = 1, \ldots, n$, and the coefficients of the polynomial $\Delta\tilde{Q}(T) = \tilde{Q}(T) - T^d$ in (8) are the solutions of $d \times d$ linear systems with a common coefficient matrix that has columns which are the coefficient vectors of*

$$(T+\Delta(T))^j \mod q(T) \quad \textit{for } j = 0, \ldots, d-1.$$

*This coefficient matrix is non-singular if and only if $u$ is a primitive element for the ideal $\langle q(T), x_1 - w_1(T), \ldots, x_n - w_n(T)\rangle$.*

*Proof.* A closer look of the definition in (7) and (8) gives the coefficient matrix of the linear systems defining $\tilde{V}_i(T)$ and $\Delta\tilde{Q}(T)$ as stated, with a common coefficient matrix. This linear system has unique solution if and only if $1, T+\Delta(T), \ldots, (T+\Delta(T))^{d-1}$ are linearly independent modulo $q(T)$, and since

$$T + \Delta(T) = \sum_{I=1}^{n} \lambda_i w_i(T) = u(\mathbf{w}(T)), \tag{10}$$

this is equivalent to $u$ being a primitive element of the updated system.

The following proposition compares Definitions 3 and 6 in cases when the coefficient ring is $R = \mathbb{Q}[t]/I^2$.

**Proposition 8.** *Assume that the conditions of Assumption 2 are satisfied and that $u$ is a primitive element for $\langle q(T), x_1 - w_1(T), \ldots, x_n - w_n(T)\rangle$ as in Proposition 7. Let $\boldsymbol{V}(T)$ and $Q(T)$ be as in Definition 3 and $\tilde{\boldsymbol{V}}(T)$ and $\tilde{Q}(T)$ be as in Definition 6 for $R = \mathbb{Q}[t]/I^2$. Then $\boldsymbol{V}(T) \equiv \tilde{\boldsymbol{V}}(T)$ and $Q(T) \equiv \tilde{Q}(T)$ mod $I^2$.*

*Proof.* From Proposition 4.(iii) we get $V_i(T+\Delta(T)) \equiv w_i(T)$ and $Q(T+\Delta(T)) \equiv 0 \mod q(T) \mod I^2$. Since $Q(T)$ is a monic polynomial of degree $d$, the coefficients of its degree $\leq d-1$ terms are uniquely determined modulo $q(T)$, so we have that

$$Q(T + \Delta(T)) - (T + \Delta(T))^d \equiv -(T + \Delta(T))^d \mod q(T) \mod I^2.$$

Since $\tilde{\boldsymbol{V}}$ and $\tilde{Q}$ are uniquely defined by these properties, they must be equal to $\boldsymbol{V}$ and $Q$ respectively.

The following proposition connects the approximate RUR defined in Definition 6 to the ones obtained by applying one step of Newton iteration on the approximate roots, as promised in the Introduction:

**Proposition 9.** *Let $\boldsymbol{F} = (F_1, \ldots, F_n) \subset \mathbb{K}[x_1, \ldots, x_n]$ as above. Assume that the polynomials $u = \lambda_1 x_1 + \cdots + \lambda_n x_n$, $q(T)$, $\mathbf{v}(T) := (v_1(T), \ldots, v_n(T))$ satisfy Assumption 5. Let $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_d \in \overline{\mathbb{K}}^n$ be the exact roots of $\langle q(T), x_1 - v_1(T), \ldots, x_n - v_n(T)\rangle \cap \mathbb{K}[x_1, \ldots, x_n]$ where $\overline{\mathbb{K}}$ is the algebraic closure of $\mathbb{K}$. Let*

$$\tilde{\boldsymbol{z}}_i := \boldsymbol{z}_i - J_{\boldsymbol{F}}(\boldsymbol{z}_i)^{-1}\boldsymbol{F}(\boldsymbol{z}_i) \quad i = 1, \ldots, d$$

*be one step of Newton iteration. Assume that $u(\tilde{\boldsymbol{z}}_i) \neq u(\tilde{\boldsymbol{z}}_j)$ for $i \neq j$. Then $\tilde{Q}(T), \tilde{V}_1(T), \ldots, \tilde{V}_n(T)$ defined in Definition 6 is the exact RUR of $\{\tilde{\boldsymbol{z}}_1, \ldots, \tilde{\boldsymbol{z}}_d\}$, with $\sum_{i=1}^{n} \lambda_i \tilde{V}_i(T) = T$.*

*Proof.* Using the notation $\boldsymbol{z}_i = (z_{i,1}, \ldots, z_{i,n})$ and $\tilde{\boldsymbol{z}}_i = (\tilde{z}_{i,1}, \ldots, \tilde{z}_{i,n})$ we define

$$\mu_i := u(\boldsymbol{z}_i) = \sum_{j=1}^{n} \lambda_j z_{i,j} \text{ and } \tilde{\mu}_i := u(\tilde{\boldsymbol{z}}_i) = \sum_{j=1}^{n} \lambda_j \tilde{z}_{i,j} \;\; i = 1, \ldots, d.$$

Then for all $i = 1, \ldots, d$ and $j = 1, \ldots, n$ we have $q(\mu_i) = 0$, $w_j(\mu_i) = \tilde{z}_{i,j}$ and $\Delta(\mu_i) = \tilde{\mu}_i - \mu_i$. Note that $u$ is a primitive element for $\langle q(T), x_1 - w_1(T), \ldots, x_n - w_n(T)\rangle$ if and only if $\tilde{\mu}_i \neq \tilde{\mu}_j$ for $i \neq j$. In this case for all $i = 1, \ldots, d$ and $j = 1, \ldots, n$ we have from (7), (8) and (9) that

$$\tilde{V}_j(\tilde{\mu}_i) = w_j(\mu_i) = \tilde{z}_{i,j} \text{ and } \tilde{Q}(\tilde{\mu}_i) = \Delta\tilde{Q}(\tilde{\mu}_i) + \tilde{\mu}_i^d = 0.$$

This proves that $\tilde{Q}(T), \tilde{V}_1(T), \ldots, \tilde{V}_n(T)$ is the exact RUR of $\{\tilde{\boldsymbol{z}}_1, \ldots, \tilde{\boldsymbol{z}}_d\}$. Finally, the last claim follows from

$$\sum_{i=1}^{n} \lambda_i \tilde{V}_i(T + \Delta(T)) = \sum_{i=1}^{n} \lambda_i w_i(T) = T + \Delta(T).$$

**Corollary 10.** *Let $R = \mathbb{K} \subset \mathbb{C}$ be a field equipped with the usual absolute value, and consider the Euclidean norm on the coefficient vectors of polynomials over $\mathbb{C}$. Then the iteration defined by Definition 6 is locally quadratically convergent to an exact RUR of a component of $\langle \boldsymbol{F} \rangle$ over an algebraic extension of $\mathbb{K}$, as long as Assumption 5 is satisfied in each iteration.*

### 3.3 Second Construction

Our second variation of Definition 3 will have the property that it can be interpreted as an $(n+1)d$ dimensional Newton iteration as follows. Given $\boldsymbol{F} = (F_1, \ldots, F_n)$ and $u = \sum_{i=1}^{n} \lambda_i x_i$ in $R[x_1, \ldots, x_n]$ as before, we define the map

$$\Phi : R^{(n+1)d} \to R^{(n+1)d}$$

as the map of the coefficient vectors of the following degree $d-1$ polynomials:

$$\Phi : \begin{bmatrix} v_1(T) \\ \vdots \\ v_n(T) \\ \Delta q(T) \end{bmatrix} \mapsto \begin{bmatrix} F_1(\mathbf{v}(T)) \bmod q(T) \\ \vdots \\ F_n(\mathbf{v}(T)) \bmod q(T) \\ \sum_{i=1}^{n} \lambda_i v_i(T) - T \end{bmatrix}, \tag{11}$$

where

$$\Delta q(T) := q(T) - T^d.$$

If $u, q(T), v_1(T), \ldots, v_n(T)$ is an exact RUR of a component of $\langle \boldsymbol{F} \rangle$ then

$$\Phi\left(v_1(T), \ldots, v_n(T), \Delta q(T)\right) = 0.$$

So one can apply the $(n+1)d$ dimensional Newton iteration to locally converge to the coefficient vector of an exact RUR which is a zero of $\Phi$. Note that below we will consider the map $\Phi$ as a map between

$$\Phi : \left(R[T]/\langle q(T)\rangle\right)^{n+1} \to \left(R[T]/\langle q(T)\rangle\right)^{n+1},$$

and note that $\left(R[T]/\langle q(T)\rangle\right)^{n+1}$ and $R^{(n+1)d}$ are isomorphic as vectors spaces when $R = \mathbb{K}$ a field. Moreover, as we will see below, the Newton iteration for $\Phi$ respects the algebra structure of $\left(R[T]/\langle q(T)\rangle\right)^{n+1}$ as well.

The first lemma gives the Jacobian matrix of $\Phi$.

**Lemma 11.** *Let $\boldsymbol{F} = (F_1, \ldots, F_n)$, $u$, $q(T)$, $\mathbf{v}(T)$ and $\varPhi$ be as above. For $i = 1, \ldots, n$ define $m_i(T)$ and $r_i(T)$ as the quotient and remainder in the division with remainder:*

$$F_i(\mathbf{v}(T)) = m_i(T)q(T) + r_i(T). \tag{12}$$

*Then the Jacobian matrix of $\varPhi$ defined in (11) and considered as a map on $(R[T]/\langle q(T)\rangle)^{n+1}$ is given by*

$$J_\varPhi(\mathbf{v}(T), \varDelta q(T)) := \begin{array}{|ccc|c|c} \multicolumn{3}{c}{n} & \multicolumn{1}{c}{1} & \\ \hline & & & -m_1(T) & \\ & J_{\boldsymbol{F}}(\mathbf{v}(T)) & & \vdots & n \\ & & & -m_n(T) & \\ \hline \lambda_1 & \cdots & \lambda_n & 0 & 1 \\ \hline \end{array} \mod q(T). \tag{13}$$

*Proof.* Using the notation $\boldsymbol{r} = (r_1(T), \ldots, r_n(T))$ from (12), we have that

$$\varPhi(\mathbf{v}(T), \varDelta q(T)) = (\boldsymbol{r}(T), \sum_{i=1}^{n} \lambda_i v_i(T)).$$

Let $v_{i,j}$ be the coefficient of $T^j$ in $v_i(T)$ for $i = 1, \ldots, n$ and $j = 0, \ldots, d-1$. Then for $k = 1, \ldots, n$

$$\begin{aligned} \frac{\partial\, r_k(T)}{\partial v_{i,j}} &= \frac{\partial F_k(\mathbf{v}(T))}{\partial v_{i,j}} - q(T)\frac{\partial m_k(T)}{\partial v_{i,j}} \\ &= \frac{\partial F_k(\mathbf{v}(T))}{\partial v_{i,j}} \mod q(T) \\ &= \frac{\partial F_k}{\partial x_i}(\mathbf{v}(T)) \cdot \frac{\partial v_i(T))}{\partial v_{i,j}} \mod q(T) \\ &= \frac{\partial F_k}{\partial x_i}(\mathbf{v}(T)) \cdot T^j \mod q(T). \end{aligned}$$

This shows that the block corresponding to the derivatives of $v_{i,j}$ for $j = 0, \ldots, d-1$ defines modular multiplication by the polynomials $\frac{\partial F_k}{\partial x_i}(\mathbf{v}(T))$, which is an entry of $J_{\boldsymbol{F}}(\mathbf{v}(T))$. Furthermore,

$$\frac{\partial \left(\sum_{k=1}^{n} \lambda_k v_k(T)\right)}{\partial v_{i,j}} = \lambda_i T^j$$

which gives the last row of (13). Let $q_j$ be he coefficient of $T^j$ in $\varDelta q(T) = q(T) - T^d$ for $j = 0, \ldots, d-1$. Then

$$\begin{aligned} \frac{\partial r_k(T)}{\partial q_j} &= \frac{\partial F_k(\mathbf{v}(T))}{\partial q_j} - q(T)\frac{\partial m_k(T)}{\partial q_j} - m_k(T)\frac{\partial q(T)}{\partial q_j} \\ &= -m_k(T)\frac{\partial q(T)}{\partial q_j} \mod q(T) \\ &= -m_k(T)T^j \mod q(T) \end{aligned}$$

and $\frac{\partial \lambda_k v_k(T)}{\partial q_j} = 0$ which gives the last column.

*Remark 12.* The proof of Lemma 11 shows that the Jacobian matrix of $\Phi$ considered as a map on $R^{(n+1)d}$ is the $(n+1)d \times (n+1)d$ matrix obtained from $J_\Phi$ in (13) by replacing every entry by the $d \times d$ matrix of multiplication modulo $q(T)$ by the polynomial in that entry. In other words, if the polynomial in the $(i, j)$-th entry of $J_\phi$ is $p(T)$ then the $k$-th column of this $d \times d$ block is the coefficient vector of $T^{k-1}p(T) \mod q(T)$.

Next, we give explicitly the iteration function corresponding to the Newton iteration on $\Phi$, using polynomial arithmetic modulo $q(T)$. We need the following assumptions:

**Assumption 13** Let $\boldsymbol{F}(x_1, \dots, x_n)$, $u = \sum_{i=1}^n \lambda_i x_i$, $q(T)$ and $\mathbf{v}(T)$ polynomials over some Euclidean domain $R$ as above. We assume that

1. $q(T)$ is monic and has degree $d$,
2. $v_i(T)$ has degree at most $d-1$,
3. $\frac{\partial q(T)}{\partial T}$ is invertible modulo $q(T)$,
4. $\lambda_1 v_1(T) + \cdots + \lambda_n v_n(T) = T$,
5. $J_{\boldsymbol{F}}(\mathbf{v}(T)) := \left[\frac{\partial F_i}{\partial x_j}(\mathbf{v}(T))\right]_{i,j=1}^n$ is invertible modulo $q(T)$.
6. $J_\Phi := J_\Phi(\mathbf{v}(T), \Delta q(T))$ defined in (13) is invertible modulo $q(T)$.

**Definition 14.** *Let $\boldsymbol{F}(x_1, \dots, x_n)$, $u(x_1, \dots, x_n)$, $q(T)$ and $\mathbf{v}(T)$ be polynomials over $R$ satisfying Assumption 13. Then we define*

$$u = \sum_{i=1}^n \lambda_i x_i = T$$

$$\boldsymbol{w}(T) := \mathbf{v}(T) - \left(J_{\boldsymbol{F}}(\mathbf{v}(T))^{-1}\boldsymbol{F}(\mathbf{v}(T)) \mod q(T)\right),$$

$$\Delta(T) := \sum_{i=1}^n \lambda_i w_i(T) - T \textit{ same as in Definitions 3 and 6,} \tag{14}$$

$$\boldsymbol{r}(T) := \boldsymbol{F}(\mathbf{v}(T)) \mod q(T) \tag{15}$$

$$\boldsymbol{U}(T) := \frac{\partial \mathbf{v}(T)}{\partial T} - \left(J_{\boldsymbol{F}}(\mathbf{v}(T))^{-1}\frac{\partial \boldsymbol{r}(T)}{\partial T} \mod q(T)\right), \tag{16}$$

$$\Lambda(T) := \sum_{i=1}^n \lambda_i U_i(T) \textit{ that we will show to be invertible modulo } q(T) \tag{17}$$

$$\bar{\boldsymbol{V}}(T) := \boldsymbol{w}(T) - \left(\frac{\Delta(T)}{\Lambda(T)}\boldsymbol{U}(T) \mod q(T)\right), \tag{18}$$

$$\bar{Q}(T) := q(T) - \left(\frac{\Delta(T)}{\Lambda(T)}\frac{\partial q(T)}{\partial T} \mod q(T)\right). \tag{19}$$

*Remark 15.* Note that in general derivation and modular arithmetic do not commute, i.e.

$$\frac{\partial p(T)}{\partial T} \mod q(T) \neq \frac{\partial (p(T) \mod q(T))}{\partial T}.$$

For example if $p = T^2$ and $q = T^2 - 1$ then the left hand side is $2T$ but the right hand side is 0. That is why we first had to introduce the reduced form of $\boldsymbol{F}(\mathbf{v}(T))$ modulo $q(T)$ in (15) and then use its derivative by $T$ in (16).

*Remark 16.* Note that if $R = \mathbb{Q}[t]$ then and $I = \langle t^k \rangle$ for some $k \geq 1$ then

$$\frac{\Delta(T)}{\Lambda(T)} \equiv \Delta(T) \text{ and } \boldsymbol{U}(T) \equiv \frac{\partial \mathbf{w}(T)}{\partial T} \mod q(T) \mod I^2,$$

thus our second construction is equivalent to the one in Definition 3. However, when our coefficient ring $R$ is a field $\mathbb{K} \subset \mathbb{C}$, the polynomials in $\boldsymbol{U}(T)$ are not the partial derivatives of the ones in $\mathbf{w}(T)$, so we get a different iteration in Definition 14 from the one in Definition 3.

The next proposition shows that $\bar{\boldsymbol{V}}(T)$ and $\bar{Q}(T)$ from Definition 14 are the Newton iterates for the function $\Phi$.

**Proposition 17.** *Let $\boldsymbol{F}$, $u$, $q(T)$, $\mathbf{v}(T)$ and $\Phi$ be such that Assumption 13 holds. Then $\Lambda(T)$ defined in (17) is invertible modulo $q(T)$, and thus $\bar{\boldsymbol{V}}(T)$ and $\bar{Q}(T)$ are well defined in Definition 14. Furthermore*

$$\begin{bmatrix} \bar{\boldsymbol{V}}(T) \\ \bar{Q}(T) - T^d \end{bmatrix} = \begin{bmatrix} \mathbf{v}(T) \\ q(T) - T^d \end{bmatrix} - J_\Phi^{-1} \cdot \begin{bmatrix} \boldsymbol{F}(\mathbf{v}(T)) \\ \sum_{i=1}^n \lambda_i v_i(T) - T \end{bmatrix} \mod q(T), \tag{20}$$

*where the vector on the right hand side is $\Phi(\mathbf{v}(T), q(T) - T^d)$. Finally, we also have that $\sum_{i=1}^n \lambda_i \bar{V}_i(T) = T$.*

*Proof.* Taking derivatives by $T$ of both sides of the equations

$$F_i(\mathbf{v}(T)) - m_i(T) q(T) = r_i(T)$$

for $i = 1, \ldots n$, we get that

$$J_{\boldsymbol{F}}(\mathbf{v}(T)) \frac{\partial \mathbf{v}(T)}{\partial T} - \frac{\partial q(T)}{\partial T} \boldsymbol{m}(T) \equiv \frac{\partial \boldsymbol{r}(T)}{\partial T} \mod q(T),$$

or equivalently

$$\begin{array}{|c|c|} \hline & -m_1(T) \\ J_{\boldsymbol{F}}(\mathbf{v}(T)) & \vdots \\ & -m_n(T) \\ \hline \end{array} \cdot \begin{bmatrix} \frac{\partial v_1(T)}{\partial T} \\ \vdots \\ \frac{\partial v_n(T)}{\partial T} \\ \frac{\partial q(T)}{\partial T} \end{bmatrix} \equiv \begin{bmatrix} \frac{\partial r_1(T)}{\partial T} \\ \vdots \\ \frac{\partial r_n(T)}{\partial T} \end{bmatrix} \mod q(T).$$

From the definition

$$\boldsymbol{U}(T) := \frac{\partial \mathbf{v}(T)}{\partial T} - \left( J_{\boldsymbol{F}}(\mathbf{v}(T))^{-1} \frac{\partial \boldsymbol{r}(T)}{\partial T} \mod q(T) \right)$$

in (16) we get that

$$\begin{array}{|c|c|} \hline & -m_1(T) \\ J_{\boldsymbol{F}}(\mathbf{v}(T)) & \vdots \\ & -m_n(T) \\ \hline \end{array} \cdot \begin{bmatrix} U_1(T) \\ \vdots \\ U_n(T) \\ \frac{\partial q(T)}{\partial T} \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix} \mod q(T). \tag{21}$$

From this and from the definition

$$\mathit{\Lambda}(T) = \sum_{i=1}^{n} \lambda_i U_i(T)$$

it is easy to see that

$$\mathit{\Lambda}(T) = \frac{\partial q(T)}{\partial T} \cdot [\lambda_1 \cdots \lambda_n]\, J_{\boldsymbol{F}}^{-1}(\mathbf{v}(T)) \begin{bmatrix} m_1(T) \\ \vdots \\ m_n(T) \end{bmatrix}.$$

Then using Schur complements we get that

$$\mathit{\Lambda}(T) \cdot \det J_{\boldsymbol{F}}(\mathbf{v}(T)) = \frac{\partial q(T)}{\partial T} \cdot \det J_{\mathit{\Phi}},$$

and since both $\frac{\partial q(T)}{\partial T}$ and $\det(J_{\mathit{\Phi}})$ are invertible modulo $q(T)$ by our assumptions, so is $\mathit{\Lambda}(T)$ as was claimed.
To prove the second claim, consider $\mathbf{w}(T) = \mathbf{v}(T) - J_{\boldsymbol{F}}(\mathbf{v}(T))^{-1}\boldsymbol{F}(\mathbf{v}(T))$ defined in (14), and define

$$\boldsymbol{W}(T) := J_{\boldsymbol{F}}(\mathbf{v}(T))^{-1}\boldsymbol{F}(\mathbf{v}(T)). \tag{22}$$

Using that $\sum_{i=1}^{n} \lambda_i v_i(T) = T$ we get that

$$\mathit{\Delta}(T) = \sum_{i=1}^{n} \lambda_i w_i(T) - T = -\sum_{i=1}^{n} \lambda_i W_i(T)$$

and thus

$$\sum_{i=1}^{n} \lambda_i W_i(T) + \frac{\mathit{\Delta}(T)}{\mathit{\Lambda}(T)} \cdot \sum_{i=1}^{n} \lambda_i U_i(T) \equiv 0 \equiv \sum_{i=1}^{n} \lambda_i v_i(T) - T \mod q(T). \tag{23}$$

Thus from (21), (22) and (23) we have that modulo $q(T)$

$$\begin{array}{|ccc|c|}\hline & & & -m_1(T) \\ & J_{\boldsymbol{F}}(\mathbf{v}(T)) & & \vdots \\ & & & -m_n(T) \\ \hline \lambda_1 & \cdots & \lambda_n & 0 \\ \hline \end{array} \cdot \begin{bmatrix} W_1(T) + \frac{\mathit{\Delta}(T)}{\mathit{\Lambda}(T)} U_1(T) \\ \vdots \\ W_n(T) + \frac{\mathit{\Delta}(T)}{\mathit{\Lambda}(T)} U_n(T) \\ \frac{\mathit{\Delta}(T)}{\mathit{\Lambda}(T)} \cdot \frac{\partial q(T)}{\partial T} \end{bmatrix} = \begin{bmatrix} F_1(\mathbf{v}(T)) \\ \vdots \\ F_n(\mathbf{v}(T)) \\ \sum_{i=1}^{n} \lambda_i v_i(T) - T \end{bmatrix},$$

or equivalently

$$\begin{bmatrix} W_1(T) + \frac{\mathit{\Delta}(T)}{\mathit{\Lambda}(T)} U_1(T) \\ \vdots \\ W_n(T) + \frac{\mathit{\Delta}(T)}{\mathit{\Lambda}(T)} U_n(T) \\ \frac{\mathit{\Delta}(T)}{\mathit{\Lambda}(T)} \cdot \frac{\partial q(T)}{\partial T} \end{bmatrix} = J_{\mathit{\Phi}}^{-1} \cdot \begin{bmatrix} F_1(\mathbf{v}(T)) \\ \vdots \\ F_n(\mathbf{v}(T)) \\ \sum_{i=1}^{n} \lambda_i v_i(T) - T \end{bmatrix}.$$

This, together with the definitions of $\bar{\boldsymbol{V}}(T)$ in (18) and $\bar{Q}(T)$ in (19) proves the second claim. The third claim is immediate from (23), as

$$\sum_{i=1}^{n} \lambda_i \bar{V}_i = \sum_{i=1}^{n} \lambda_i v_i(T) - \sum_{i=1}^{n} \lambda_i \left( W_i(T) + \frac{\Delta(T)}{\Lambda(T)} U_i(T) \right) = T - 0.$$

**Corollary 18.** *Let $R = \mathbb{K} \subset \mathbb{C}$ be a field equipped with the usual absolute value, and consider the Euclidean norm on the coefficient vectors of polynomials over $\mathbb{C}$. Then the iteration defined by Definition 14 is locally quadratically convergent to an exact RUR of a component of $\langle \boldsymbol{F} \rangle$ over an algebraic extension of $\mathbb{K}$, as long as Assumption 13 is satisfied in each iteration.*

## 4 Example: A cubic-centered 12-bar linkage

To illustrate the application of these techniques, we compute an RUR for a rational component of a square system to prove that it solves an overdetermined system of equations arising from a 12-bar spherical linkage. The overdetermined polynomial system $\boldsymbol{G}$ consists of 17 quadratic and 2 linear polynomials in 18 variables for the linkage first described in [40], which is presented in [39, Fig. 3]. The trivial rotation of the cube is removed by placing the center of the cube at the origin and fixing two adjacent vertices, say $P_7$ and $P_8$, at $(-1, 1, -1)$ and $(-1, -1, -1)$, respectively. The 18 variables are the coordinates of the six remaining vertices of the cube, say $P_1, \ldots, P_6$ with $P_i = (P_{ix}, P_{iy}, P_{iz})$. The 17 quadratic conditions force these free vertices to maintain their relative distances:

$$\|P_i - P_j\|^2 - 4 = 0, \, (i,j) \in \left\{ \begin{array}{c} (1,2),(3,4),(5,6),(1,5),(2,6),(3,7), \\ (4,8),(1,3),(2,4),(5,7),(6,8) \end{array} \right\}$$

$$\|P_i\|^2 - 3 = 0, \qquad i = 1, \ldots, 6.$$

The irreducible components of these 17 quadratic polynomials was first described in [17, Table 1]. This decomposition shows that there is a unique irreducible surface $S$ of degree 16, which is the current focus of study. In particular, the rational component is the 16 points arising from the intersection of $S$ with the codimension two linear space defined by:

$$\begin{array}{l} P_{3x} + P_{4x} + P_{2z} = 0, \\ P_{5x} - P_{6x} + P_{3y} + P_{3z} + 1 = 0. \end{array}$$

In order to compute an RUR for this rational component, we consider the square polynomial system $\boldsymbol{F}$ consisting of these two linear equations and 16 quadratic equations obtained by adding the last quadratic above, i.e., $\|P_6\|^2 - 3$, to the other sixteen. Starting with a witness set for $S$, we used `Bertini` [4] to compute numerical approximations of the 16 points of interest. From these points, we observe that the variable $P_{6z}$ is distinct, so we take the primitive element $u(P_1, \ldots, P_6) = P_{6z} = T$. Next, we produce an initial guess for the monic univariate polynomial $q(T)$ of degree 16. Since $q(T)$ naturally has small integer coefficients, this polynomial was computed exactly. We produced an initial guess for the univariate polynomials $\mathbf{v}(T)$ of degree at most 15 via Lagrange interpolation using the computed numerical approximations. These are polynomials with rational coefficients having at most 5 digit numerators and denominators. These initial guesses are:

$$\begin{aligned}
q(T) &= T^{16} + 20T^{15} + 210T^{14} + 1230T^{13} + 4212T^{12} + 4677T^{11} \\
&\quad - 6886T^{10} - 21389T^{9} + 58242T^{8} - 45269T^{7} - 6118T^{6} \\
&\quad + 58968T^{5} - 103014T^{4} + 119847T^{3} - 91281T^{2} + 40466T - 8291 \\
v_{1x}(T) &= -1 \\
v_{2x}(T) &= -1 \\
v_{3x}(T) &= -1/1112T^{15} - 1/57T^{14} - 13/72T^{13} - 139/138T^{12} - 167/54T^{11} \\
&\quad - 67/60T^{10} + 1616/117T^{9} + 3173/120T^{8} - 3922/63T^{7} \\
&\quad + 1165/39T^{6} + 2909/123T^{5} - 1709/30T^{4} + 9965/109T^{3} \\
&\quad - 2366/23T^{2} + 63032/1027T - 1466/99 \\
v_{4x}(T) &= 1/1112T^{15} + 1/57T^{14} + 13/72T^{13} + 139/138T^{12} + 167/54T^{11} \\
&\quad + 67/60T^{10} - 1616/117T^{9} - 3173/120T^{8} + 3922/63T^{7} \\
&\quad - 1165/39T^{6} - 2909/123T^{5} + 1709/30T^{4} - 9965/109T^{3} \\
&\quad + 2366/23T^{2} - 63032/1027T + 1565/99 \\
v_{5x}(T) &= -1/10090T^{15} - 1/506T^{14} - 1/49T^{13} - 3/26T^{12} - 33/94T^{11} \\
&\quad - 7/132T^{10} + 187/83T^{9} + 538/107T^{8} - 398/77T^{7} \\
&\quad - 1601/587T^{6} + 2223/502T^{5} - 129/113T^{4} + 544/63T^{3} \\
&\quad - 917/150T^{2} - 233/97T + 23/36 \\
v_{6x}(T) &= -1/5519T^{15} - 1/265T^{14} - 3/73T^{13} - 31/121T^{12} - 33/34T^{11} \\
&\quad - 129/79T^{10} - 37/517T^{9} + 233/61T^{8} - 2017/270T^{7} \\
&\quad + 373/177T^{6} + 155/53T^{5} - 2314/271T^{4} + 705/59T^{3} \\
&\quad - 4487/358T^{2} + 963/158T - 59/47 \\
v_{1y}(T) &= 1 \\
v_{2y}(T) &= -1 \\
v_{3y}(T) &= -1/2038T^{15} - 1/103T^{14} - 17/169T^{13} - 31/54T^{12} - 245/132T^{11} \\
&\quad - 120/89T^{10} + 224/39T^{9} + 757/60T^{8} - 581/18T^{7} \\
&\quad + 3193/184T^{6} + 1661/150T^{5} - 21047/654T^{4} + 7721/163T^{3} \\
&\quad - 7541/138T^{2} + 2725/78T - 785/94 \\
v_{4y}(T) &= -1/2357T^{15} - 1/108T^{14} - 33/314T^{13} - 39/56T^{12} - 187/65T^{11} \\
&\quad - 581/95T^{10} - 2401/600T^{9} + 930/101T^{8} - 775/113T^{7} \\
&\quad - 445/44T^{6} + 1237/142T^{5} - 1487/168T^{4} + 1817/150T^{3} \\
&\quad - 462/167T^{2} - 1489/248T + 494/135 \\
v_{5y}(T) &= 1/12979T^{15} + 1/694T^{14} + 1/70T^{13} + 11/148T^{12} + 37/177T^{11} \\
&\quad + 1/107T^{10} - 213/365T^{9} + 118/195T^{8} + 1551/163T^{7} \\
&\quad - 580/67T^{6} - 531/166T^{5} + 1893/211T^{4} - 463/51T^{3} \\
&\quad + 1090/61T^{2} - 2787/212T + 258/95 \\
v_{6y}(T) &= 1/5519T^{15} + 1/265T^{14} + 3/73T^{13} + 31/121T^{12} + 33/34T^{11} \\
&\quad + 129/79T^{10} + 37/517T^{9} - 233/61T^{8} + 2017/270T^{7} \\
&\quad - 373/177T^{6} - 155/53T^{5} + 2314/271T^{4} - 705/59T^{3} \\
&\quad + 4487/358T^{2} - 1121/158T + 12/47 \\
v_{1z}(T) &= -1 \\
v_{2z}(T) &= -1 \\
v_{3z}(T) &= 1/2448T^{15} + 1/126T^{14} + 2/25T^{13} + 13/30T^{12} + 183/148T^{11} \\
&\quad - 22/95T^{10} - 1299/161T^{9} - 553/40T^{8} + 1259/42T^{7} \\
&\quad - 338/27T^{6} - 3182/253T^{5} + 1958/79T^{4} - 1630/37T^{3} \\
&\quad + 4292/89T^{2} - 1084/41T + 1206/221 \\
v_{4z}(T) &= -1/2106T^{15} - 1/119T^{14} - 4/53T^{13} - 23/74T^{12} - 25/116T^{11} \\
&\quad + 5654/1131T^{10} + 2868/161T^{9} + 1327/77T^{8} - 7423/134T^{7} \\
&\quad + 2759/69T^{6} + 1718/115T^{5} - 9575/199T^{4} + 5393/68T^{3} \\
&\quad - 12613/126T^{2} + 6401/95T - 1883/92 \\
v_{5z}(T) &= 1/5677T^{15} + 1/292T^{14} + 7/202T^{13} + 26/137T^{12} + 219/391T^{11} \\
&\quad + 22/353T^{10} - 139/49T^{9} - 774/175T^{8} + 279/19T^{7} \\
&\quad - 587/99T^{6} - 450/59T^{5} + 536/53T^{4} - 3029/171T^{3} \\
&\quad + 1343/56T^{2} - 462/43T + 14/13 \\
v_{6z}(T) &= T
\end{aligned}$$

We refined the approximate RUR using Algorithm 19, and in 3 iterations (in roughly 1 second) we found the exact RUR. In the $\mathbf{v}(T)$ polynomials of the exact RUR, the numerators and denominators of the coefficients have at most 28 digits, namely:

$$
\begin{aligned}
\alpha \quad &= 1/3204471773221369279790658525 \\
v_{3x}(T) &= \alpha(-2881129493593630865610329T^{15} - 56469358709164889119641644T^{14} \\
&\quad - 578442048083015317390422659T^{13} - 3227775460749576025678391459T^{12} \\
&\quad - 9909894946587188228883719582T^{11} - 3578358749346900975113448620T^{10} \\
&\quad + 44260151084205755500190960589T^{9} + 84731577601881128711018565420T^{8} \\
&\quad - 199491165378780802464515305188T^{7} + 95723229838339681423971314578T^{6} \\
&\quad + 75787130941751596487093105995T^{5} - 182548470032615020420523374937T^{4} \\
&\quad + 292959497003500175534452099849T^{3} - 329643042476857281605069314889T^{2} \\
&\quad + 196674125364601362085119810025T - 47452126308845628915580789974) \\
v_{4x}(T) &= \alpha(2881129493593630865610329T^{15} + 56469358709164889119641644T^{14} \\
&\quad + 578442048083015317390422659T^{13} + 3227775460749576025678391459T^{12} \\
&\quad + 9909894946587188228883719582T^{11} + 3578358749346900975113448620T^{10} \\
&\quad - 44260151084205755500190960589T^{9} - 84731577601881128711018565420T^{8} \\
&\quad + 199491165378780802464515305188T^{7} - 95723229838339681423971314578T^{6} \\
&\quad - 75787130941751596487093105995T^{5} + 182548470032615020420523374937T^{4} \\
&\quad - 292959497003500175534452099849T^{3} + 329643042476857281605069314889T^{2} \\
&\quad - 196674125364601362085119810025T + 50656598082066998195371448499) \\
v_{5x}(T) &= \alpha/47(-14926508985814692725660295T^{15} - 297773391799482191643772465T^{14} \\
&\quad - 3081610308415501202687193085T^{13} - 17389506895894299310438310140T^{12} \\
&\quad - 52870530884177916409158871660T^{11} - 7985554397153232406118329735T^{10} \\
&\quad + 339337071563230521110991064610T^{9} + 757269429739682916272675120805T^{8} \\
&\quad - 778491818056744498922090610960T^{7} - 410778307074659870607871725630T^{6} \\
&\quad + 666945252290852928200793739220T^{5} - 171936037274621562789031991275T^{4} \\
&\quad + 1300517229936781609262733002915T^{3} - 920730755436417716152608059995T^{2} \\
&\quad - 361767415638579519536847568045T + 96208631064260278520898204560) \\
v_{6x}(T) &= \alpha/47(-27287280868886188971151890T^{15} - 567883140744448436140744235T^{14} \\
&\quad - 6190772706188356762139486930T^{13} - 38580721830051962511624944735T^{12} \\
&\quad - 146182732583672120750472401420T^{11} - 245939987210083103698877745635T^{10} \\
&\quad - 10778669490043487418881186945T^{9} + 575272052935709136436375750005T^{8} \\
&\quad - 1125114039605827435077357466500T^{7} + 317389390835062387309791675960T^{6} \\
&\quad + 440466388670191221545685003445T^{5} - 1286022849359951198330786683160T^{4} \\
&\quad + 1799659504525177884513418973560T^{3} - 1887676569404811089807938408490T^{2} \\
&\quad + 917961856537764412658257692880T - 189055429157249714249696125460) \\
v_{3y}(T) &= \alpha(-1572062320020980286480607T^{15} - 31108187322082085458724777T^{14} \\
&\quad - 322297219762495483795767647T^{13} - 1839326187121146132000755672T^{12} \\
&\quad - 5947630470096723947860088831T^{11} - 4320609510981215075777569160T^{10} \\
&\quad + 18405439998982941489139392512T^{9} + 40429646494515311378527374510T^{8} \\
&\quad - 103433053131401921829654108504T^{7} + 55608079790549864732386331774T^{6} \\
&\quad + 35484215857890035406790077085T^{5} - 103126222378917399950067588446T^{4} \\
&\quad + 151789772699498984312446113442T^{3} - 175108178833837080947804129237T^{2} \\
&\quad + 111951203875666042023358897150T - 26760787199332495231945653817) \\
v_{4y}(T) &= \alpha(-1359749822277659136367732T^{15} - 29651222724814187524939277T^{14} \\
&\quad - 336783507299580007515911147T^{13} - 2231865653484431345450780072T^{12} \\
&\quad - 9219287203307410697519852906T^{11} - 19597877374389843525839721035T^{10} \\
&\quad - 12823225261355870172371961013T^{9} + 29506643456688254696406280185T^{8} \\
&\quad - 21977689644205909107128376729T^{7} - 32409171956862700073069418826T^{6} \\
&\quad + 27915029062394568996017596135T^{5} - 28363400298198409791080814221T^{4} \\
&\quad + 38816863323840423843221276617T^{3} - 8865121893390327106993539437T^{2} \\
&\quad - 19239716389698007639434041750T + 11725943590084476919258204283) \\
v_{5y}(T) &= \alpha/47(11603983966259833376058160T^{15} + 217153170339381974795502905T^{14} \\
&\quad + 2139063241115082817328441135T^{13} + 11191289970084093849864343715T^{12} \\
&\quad + 31486473569992118869518659780T^{11} + 1401302126019324290665142660T^{10} \\
&\quad - 87890587988818445033654819495T^{9} + 91139458508898945030483080370T^{8} \\
&\quad + 1433108663902588131761054475525T^{7} - 1303792568225360896022867456000T^{6} \\
&\quad - 481771480980237531202595548510T^{5} + 1351208978813262782720191318100T^{4} \\
&\quad - 1367308841574912038190287285515T^{3} + 2691219896341123859883175304435T^{2} \\
&\quad - 1979954015860581566552572962955T + 409019746222899610159043835865) \\
v_{6y}(T) &= \alpha/47(27287280868886188971151890T^{15} + 567883140744448436140744235T^{14} \\
&\quad + 6190772706188356762139486930T^{13} + 38580721830051962511624944735T^{12} \\
&\quad + 146182732583672120750472401420T^{11} + 245939987210083103698877745635T^{10} \\
&\quad + 10778669490043487418881186945T^{9} - 575272052935709136436375750005T^{8} \\
&\quad + 1125114039605827435077357466500T^{7} - 317389390835062387309791675960T^{6} \\
&\quad - 440466388670191221545685003445T^{5} + 1286022849359951198330786683160T^{4} \\
&\quad - 1799659504525177884513418973560T^{3} + 1887676569404811089807938408490T^{2} \\
&\quad - 1068572029879168768808418643555T + 38445255815845358099535174785) \\
v_{3z}(T) &= \alpha(1309067173572650579129722T^{15} + 25361171387082803660916867T^{14} \\
&\quad + 256144828320519833594655012T^{13} + 1388449273628429893677635787T^{12} \\
&\quad + 3962264476490464281023630751T^{11} - 742250761634314100664120540T^{10} \\
&\quad - 25854711085222814011051568077T^{9} - 44301931107365817332491190910T^{8} \\
&\quad + 96058112247378880634861196684T^{7} - 40115150047789816691584982804T^{6} \\
&\quad - 40302915083861561080303028910T^{5} + 79422247653697620470455786491T^{4} \\
&\quad - 141169724304001191222005986407T^{3} + 154534863643020200657265185652T^{2} \\
&\quad - 84722921488935320061760912875T + 17486867336291764403844477632) \\
v_{4z}(T) &= \alpha(-1521379671315971729242597T^{15} - 26818135984350701594702367T^{14} \\
&\quad - 241658540783435309874511512T^{13} - 995909807265144680227611387T^{12} \\
&\quad - 690607743279777531363866676T^{11} + 16019518625042942550726272415T^{10} \\
&\quad + 57083376345561625672562921602T^{9} + 55224934145192874014612285235T^{8} \\
&\quad - 177513475734574893357386928459T^{7} + 128132401795202381497040733404T^{6} \\
&\quad + 47872101879357027491075509860T^{5} - 154185069734416610629442560716T^{4} \\
&\quad + 254142633679659751691230823232T^{3} - 320777920583466954498075775452T^{2} \\
&\quad + 215913841754299369724553851775T - 65587013445372844394420311307) \\
v_{5z}(T) &= \alpha/47(26530492952074526101718455T^{15} + 514926562138864166439275370T^{14} \\
&\quad + 5220673549530584020015634220T^{13} + 28580796865978393160302653855T^{12} \\
&\quad + 84357004454170035278677531440T^{11} + 9386856523172556696783472395T^{10} \\
&\quad - 427227659552048966144645884105T^{9} - 666129971230783971242192040435T^{8} \\
&\quad + 2211600481959332630683145086485T^{7} - 893014261150701025414995730370T^{6} \\
&\quad - 1148716733271090459403389287730T^{5} + 1523145016087884345509223309375T^{4} \\
&\quad - 2667826071511693647453020288430T^{3} + 3611950651777541576035783364430T^{2} \\
&\quad - 1618186600222002047015725394910T + 162200941817234975487984680630)
\end{aligned}
$$

We checked $\boldsymbol{F}(\mathbf{v}(T)) \equiv 0 \mod q(T)$ and $\boldsymbol{G}(\mathbf{v}(T)) \equiv 0 \mod q(T)$ using exact arithmetics, meaning that we found an exact rational component of our zero dimensional ideal and proves it satisfies the overdetermined system of equations.

## 5 Parallel complexity

In this section, we study two parallel algorithms for our two constructions in Definitions 6 and 14, and analyze their parallel complexity. We express our complexity results as functions of the number of variables $n$ and the number of roots $d$. In many applications, the number of roots $d$ is large, possibly being an exponential function of $n$. Our goal is to demonstrate that we can efficiently distribute our computations to polynomially many processors in $n$ and $d$ so that the parallel computational time is polynomial in $\log(d)$ and $n$.

Note that for large $d$, the bottleneck of the computation of the formulas of Definitions 6 and 14 is the parallel computation of modular inverses of polynomials modulo the degree $d$ polynomial $q(T)$. We consider two approaches to compute modular inverses, and to modular arithmetic in general:

1. Via the computation of the roots of the modulus $q(T)$ and using parallel interpolation algorithms at these roots.
2. Via the parallel solution of Toeplitz-like linear systems.

Our first construction in Definition 6 is particularly well suited for the approach via the roots of the modulus $q(T)$ due to Proposition 9. In fact, the roots of $q(T)$ are combinations of the coordinates of approximate roots of a component of the input system $\boldsymbol{F} = (F_1, \ldots, F_n)$, and according to Proposition 9, the polynomials $\tilde{Q}(T), \tilde{\boldsymbol{V}}(T) = (\tilde{V}_1(T), \ldots, \tilde{V}_n(T))$ in Definitions 6 can be obtained from the coordinates of higher accuracy roots by using the interpolation formulas in (4) and (5). Thus, we propose to compute one iteration for our first construction by using the underlying local data and interpolation. Below, in Subsection 5.2, we analyze the parallel complexity of this algorithm.

For our second construction in Definition 14, unfortunately we do not have any results to connect the roots of the modulus with the common roots of the input $\boldsymbol{F} = (F_1, \ldots, F_n)$. In theory, we could approximate the roots of our modulus $q(T)$ in each iteration. However, to obtain exact modular arithmetics modulo $q(T)$ using these approximate roots would require further considerations, which we bypass in this paper, offering instead the algorithm for our first construction outlined above.

Instead, for our second construction we propose the parallel solution of a general Toeplitz-like linear system of equations, which refines and modifies the algorithm of [30] by using a more efficient displacement representation with factor circulant matrices, defined in [31, Example 4.4.2], rather than triangular Toeplitz matrices. This improvement will reduce the number of required FFT computations by a factor of 2. Below, in Subsection 5.3, we detail this new modified algorithm and analyze its parallel complexity. Then, we apply these complexity bounds for the computation of modular inverses and for the computation of the polynomials in Definition 14. Finally, we briefly discuss the computation of modular inverses (or rather cofactors) when all the roots of at least one of the two associated input polynomials are simple and known.

### 5.1 Computational model

We have weighed the possibility of using several parallel computational models in our complexity estimates, including message passing interface (MPI) that measures communication costs related to passing data, and PRAM (Parallel Random Access Machine) models that assume that data is readily available to all processors via shared memory.

From a computational standpoint, the current “gold standard” is a hybrid model combining both message passing and shared memory. For example, a realistic cluster (the one that we used) consists of 13 nodes with each node having 64 processors that share memory – one would like parallel programming to send data between the nodes that is then shared amongst all the local processors.

Beyond choosing between different architectures, we also have to decide between an algebraic or a Boolean computational model. The algebraic model assumes that the basic arithmetic operations in the coefficient field can be done at unit cost, independently of the size of the numbers appearing in the computation, while under the Boolean model the estimates depend on the size of the numbers as well.

In our applications, the iterations under study will be conducted using floating point complex numbers as coefficients. However, for the purposes of the complexity analysis, in each iterations we assume exact rational arithmetic using the input floating point numbers as exact rational numbers, and at the end of the iteration we round them to the desired precision. Alternatively, we can round the numbers after each arithmetic operations to the desired precision. Thus, up to a constant multiple that depends on the desired precision, we can assume that at each iteration we are dealing with rational numbers or Gaussian rationals, and arithmetics on them has unit cost (cf. [6]). That is why we choose the algebraic computational model and not the Boolean one to estimate the parallel complexity of our iterations.

Ultimately we choose the PRAM arithmetic model [26], in which we more conveniently expose our complexity estimates, but we can readily obtain from our algorithms similar estimates in terms of basic operations (like FFT), which are efficient under any reasonable model. We will invoke Brent’s scheduling principle that allows us to save processors by slowing down the computations, so that $O_A(t,p)$ will denote the simultaneous upper bounds $O(ts)$ on the parallel arithmetic time, and $\lceil p/s \rceil$ on the number of processors involved, where any $s \geq 1$ can be assumed.

### 5.2 Parallel complexity of the first construction from the roots

By Proposition 9, the iterates of Definition 6 are the same as the Lagrange interpolants of the approximate roots obtained from one step local Newton iteration. We assume here that the coordinates of the approximate roots are given as floating point complex numbers, thus our base field is $\mathbb{K} := \mathbb{Q}(i)$. Below we give estimates on the parallel complexity of the following simple algorithm:

**Algorithm 19** Computation of RUR from roots.

```
INPUT: A primitive element $u = \lambda_1 x_1 + \ldots + \lambda_n x_n$ and approximate roots
    $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_d \in \mathbb{K}^n$. We assume that the corresponding RUR $u, q(T), \mathbf{v}(T)$ satisfies Assumption 5 (but need not to be given explicitly).
OUTPUT: The updated RUR $\tilde{Q}(T), \tilde{\boldsymbol{V}}(T)$ defined in Definition 6, and its common roots $\tilde{\boldsymbol{z}}_1, \ldots, \tilde{\boldsymbol{z}}_d \in \mathbb{K}^n$.
COMPUTATIONS:
    1. Compute $\tilde{\boldsymbol{z}}_i := \boldsymbol{z}_i - J_{\boldsymbol{F}}(\boldsymbol{z}_i)^{-1}\boldsymbol{F}(\boldsymbol{z}_i) \quad i = 1, \ldots, d$
    2. Compute $\tilde{Q}(T) := \prod_{i=1}^{d}(T - u(\tilde{\boldsymbol{z}}_i))$
    3. Interpolate the polynomials $\tilde{V}_1(T), \ldots, \tilde{V}_n(T)$ such that $\tilde{V}_j(u(\tilde{\boldsymbol{z}}_i)) = \tilde{z}_{i,j}$
       for $i = 1, \ldots, d$ and $j = 1, \ldots n$.
```

The next proposition gives the complexity bounds for Algorithm 19. In what follows we use the following notation:

- $\log^{(0)}(N) := N$, $\log^{(h)}(N) := \log_2(\log^{(h-1)}(N))$, for $h \geq 1$, and $\log^*(N) := \max\{h : \ \log^{(h)}(N) > 0\}$;
- We use the exponent $\omega$ to denote a number such that $O(n^\omega)$ arithmetic operations are sufficient for the multiplication of two $n \times n$ matrices. Note that $2 \leq \omega \leq 2.373$.

**Proposition 20.** *Given $u = \sum_{i=1}^{n} \lambda_i x_i$, and $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_d \in \mathbb{K}^n$ satisfying the assumptions of Algorithm 19. Then, we can compute the polynomials $\tilde{Q}(T), \tilde{\boldsymbol{V}}(T)$ of Definition 6 and the corresponding approximate roots $\tilde{\boldsymbol{z}}_1, \ldots, \tilde{\boldsymbol{z}}_d \in \mathbb{K}^n$ of $\boldsymbol{F}$ in two stages with respective costs*

$$O_A(\log^2(n), n^{\omega+1}) \text{ and } O_A(\log^2(d)\log^*(nd), nd/\log^*(d)).$$

*Proof.* The $O_A(\log^2(n), n^{\omega+1})$ term comes from Step 1 using [5, page 319]. In Step 2 and 3 we apply the multipoint polynomial evaluation and polynomial interpolation algorithms in Sec 3.1, 3.3, and 3.13 of [31]. These algorithms essentially amount to $O(\log(d))$ steps each performing concurrent univariate multiplications and divisions of polynomials having degrees at most $d$. We can perform these operations in time $O(\log^2(d))$ using $O(nd/\log^*(d))$ processors.

### 5.3 Parallel complexity of the second construction using modular arithmetics

In this subsection, we analyze the parallel complexity of one iteration defined in the polynomial arithmetics modulo $q(T)$ in Definition 14.

In Definition 14, we have to compute two modular inverses:

$$\frac{1}{\det J_{\boldsymbol{F}}(\boldsymbol{v}(T))} \text{ and } \frac{1}{\Lambda(T)} \mod q(T),$$

which are the bottleneck of the computation if $d = \deg q(T)$ is large in comparison to $n$. One way to compute modular inverses is via the solution of a non-singular linear system with Sylvester coefficient matrix, which is a Toeplitz-like structured matrix.

In what follows, we describe an algorithm for the solution of a general Toeplitz-like linear system of equations, which refines the algorithm of [30] with more efficient displacement representation of [31, Example 4.4.2] using factor circulant matrices. Then, we briefly cover the special case where all roots of the input polynomials or at least one of them are simple and available, in which case even more efficient algorithms can be applied.

**Parametrized Newton's iteration.** Hereafter $I_m$ denotes the $m \times m$ identity matrix. We begin with recalling parametrized Newton's iteration that computes a sequence of the powers of a matrix.

**Algorithm 21** Parametrized Newton's iteration.

Input: *two positive integers $k$ and $m$ and an $m \times m$ matrix $B$.*
Output: *the powers $I_m = B^0$, $B$, $B^2, \ldots,$ $B^k$, defined as the coefficients of the matrix polynomial $A^{-1} \mod \lambda^{k+1}$ for $A = I_m - \lambda B$.*
Initialization: *Set $X_0 \leftarrow I_m$, $A \leftarrow I_m - \lambda B$ for a scalar parameter $\lambda$, $p \leftarrow \lceil \log_2(k+1) \rceil$.*
Computations:
**Stage i**, *$i = 1, \ldots, p$. Compute $X_i = X_{i-1}(2I_m - AX_{i-1})$, the matrix polynomial in $\lambda$. Output the matrix polynomial $X_p \mod \lambda^{k+1}$.*

Paper [30] first shows that $X_i = A^{-1} \mod \lambda^{2^i} = \sum_{j=0}^{2^i-1} (\lambda B)^j$ for all $i$ over any ring of constants, then proves correctness of the algorithm, and finally extends it to solving a nonsingular linear system $B\mathbf{y} = \mathbf{f}$ as follows.

**Algorithm 22** Extension to LIN·SOLVE.

Input: *two positive integers $m$, a vector $\mathbf{f}$ of dimension $m$, and the powers $I_m = B^0$, $B$, $B^2, \ldots,$ $B^m$ of a nonsingular $m \times m$ matrix $B$, defined as the coefficients of the matrix polynomial $A^{-1} \mod \lambda^{m+1}$ for $A = I_m - \lambda B$.*
Output: *the vector $\mathbf{y} = B^{-1}\mathbf{f}$.*
Computations:

1. *Compute the traces of the matrices $B^i$, $i = 1, 2, \ldots, m$ as the coefficients of the trace of the matrix $A^{-1} \mod \lambda^{m+1}$, which is a matrix polynomial in $\lambda$.*
2. *Compute the coefficients $c_0, \ldots, c_{m-1}$ of the characteristic polynomial $c_B = \sum_{j=0}^{m} c_i \lambda^i = \det(\lambda I_m - B)$*
3. *Note that $c_0 \neq 0$ because the matrix $B$ is assumed to be nonsingular, write $c_m = 1$, and compute and output the vector $\mathbf{y} = B^{-1}\mathbf{f} = -\sum_{j=0}^{m} (c_i/c_0) B^{i-1}\mathbf{f}$.*

There are more efficient parallel algorithms for the solution of a general linear system of equations, but next we refine it to obtain a superior parallel algorithm in the case of Toeplitz-like matrices $B$. We first recall some relevant definitions.

**Toeplitz-like matrices: fundamentals.** Write $J_1 = (1)$ is a $1 \times 1$ matrix, $J_k = \begin{pmatrix} \mathbf{0}^T & 1 \\ J_{k-1} & \mathbf{0} \end{pmatrix}$ for $k = 2, 3, \dots, m$, $J = J_m$ is the $m \times m$ reflection matrix. $Z_f = \begin{pmatrix} \mathbf{0}^T & f \\ I_{m-1} & \mathbf{0} \end{pmatrix}$ denotes the $n \times n$ matrix of the $f$-circular shift for a scalar $f \neq 0$.

$Z_f(\mathbf{v}) = \sum_{i=0}^{m-1} v_i Z_f^i$ is an $f$-circulant matrix, defined by its first column $\mathbf{v} = (v_i)_{i=0}^{m-1}$ and a scalar $f \neq 0$ and called circulant for $f = 1$. These matrices belong to the class of Toeplitz matrices $[t_{i-j}]_{i,j=0}^{m-1}$, which in turn can be extended to the class $\mathcal{T}$ of Toeplitz-like matrices. The *Sylvester displacements* $Z_e T - T Z_f$ of a Toeplitz-like matrix $T$ has small ranks (meant "small" in context) for a fixed pair of distinct scalars $e$ and $f$, e.g., $e = 1$, $f = -1$. These ranks are called *displacement ranks.* In particular Toeplitz, Sylvester and Frobenius companion matrices of any size have displacement ranks at most 2.

Recall that an $m \times m$ matrix $M$ of a rank at most $r$ can be expressed nonuniquely through its *generator* $(G, H)$ of length $r$,

$$M = \sum_{i=1}^{d} \mathbf{g}_i \mathbf{h}_i^T = GH^T \tag{24}$$

where $G = (\mathbf{g}_1 \ \cdots \ \mathbf{g}_r)$ and $H = (\mathbf{h}_1 \ \cdots \ \mathbf{h}_r)$ and call a generator for a displacement of a matrix a *displacement generator* for the matrix itself.

**Theorem 23.** *(See [31, Example 4.4.2].) Assume a displacement generator $(G, H)$ of a length $r$ in (24) for a matrix $Z_e T - T Z_f$ where $e \neq f$. Then this matrix can be expressed as follows: $(e - f)T = \sum_{i=1}^{r} dZ_e(\mathbf{g}_i) Z_f(J\mathbf{h}_i)^T$.*

The latter compressed representation of an $m \times m$ matrix $T$ uses $2rm$ parameters rather than $m^2$ entries and enables fast multiplication of the matrix by a vector if $m \gg r$. Indeed, the theorem reduces this operation to $r$ concurrent multiplications of $e$-circulant matrices by vectors, followed by $r$ such multiplications by $f$-circulant matrices, and finally to the summation of $r$ vectors. Next, we estimate the computational cost of such a multiplication.

**Theorem 24.** *(See [31, equations (2.4.3) and (2.4.4) and Theorem 2.6.4].)*

*(i) Multiplication of an $m \times m$ Toeplitz matrix by a vector can be reduced to multiplication of two univariate polynomials of degrees $3m - 3$ and $m - 1$.*
*(ii) Multiplication of an $f$-circulant matrix of size $m \times m$ by a vector can be reduced to performing three Fourier transforms at $m$ points and to four multiplications of vectors, one by a scalar $1/m$ and three other ones by three diagonal matrices of size $m \times m$. Two of these matrices turn into the identity matrix $I_m$ if $f = 1$.*

**The complexity of Newton's iteration for Toeplitz-like matrices.** We immediately verify the following result.

**Theorem 25.** *(See [31, Theorem 1.5.3].) Let $M$ be a nonsingular $m \times m$ matrix. Then $VM^{-1} - M^{-1}U = -M^{-1}(UM - MV)M^{-1}$ for any pair of $m \times m$ operator matrices $U$ and $V$, and so if $UM - MV = GH^T$, then $VM^{-1} - M^{-1}U = G_-H_-^T$ where $G_- = -M^{-1}G$ and $H_-^T = H^TM^{-1}$.*

Substitute $U = Z_e$ and $V = Z_f$ and deduce that the inversion of a nonsingular matrix given with its displacement does not change the length of this generator (and consequently does not change the displacement rank), provided that we reverse the order for the pair of operator matrices $(Z_e, Z_f)$, replacing it by the pair $(Z_f, Z_e)$. This observation motivates our search for a short displacement generator of the inverse of a Toeplitz-like matrix. Having such generator available, we can readily compute the solution $\mathbf{y} = T^{-1}\mathbf{f}$ of the linear system $T\mathbf{y} = \mathbf{f}$ by applying the algorithms that support Theorem 23.

We can assume that short displacement generators for the input matrix and for $X_0 = I_m$ are given (note that $Z_eI - IZ_f = Z_e - Z_f = (e - f)\mathbf{i}_1\mathbf{i}_m^T$ where $\mathbf{i}_h$ is the $h$th coordinate vector), and recursively compute short displacement generators of the matrices $X_{i+1} = A^{-1} \mod \lambda^{2^{i+1}}$ for $i = 0, 1, \ldots, p-1$ by applying the following result.

**Theorem 26.** *Assume that we are given a nonsingular $m \times m$ matrix $M$, the matrices $X_i$, $i = 0, 1, \ldots, m$ of Algorithm 21, and any pair of $m \times m$ operator matrices $U$ and $V$. Let $UM - MV = GH^T$ for some $m \times r$ matrices $G$ and $H$. Then, $VX_i - X_iU = G_iH_i^T$ where $G_i = -X_iG$ and $H_i^T = H^TX_i$ for all $i$.*

*Proof.* The theorem follows from Theorem 25 applied modulo $\lambda^{2^i}$ to the matrices $M = A$ and $M^{-1} = A^{-1} = X_i$.

We assume $m \times r$ matrices $G$ and $H$ of a displacement generator for the input matrix $A$, and so for every $i$, multiplication of each of these matrices by the matrix $X_i = X_{i-1}(2I - AX_{i-1})$ amounts to concurrent multiplication of the matrix by $r$ vectors. To operate with the matrices $X_{i-1}$ we recursively define their displacement generators of length at most $r$ and then employ the algorithms supporting Theorem 24. It follows that for every $i$, $i = 1, \ldots, p$ we compute a short generator of the matrix $X_i$ at the computational cost dominated by the cost of performing $O(r^2)$ multiplications of bivariate polynomials of degrees at most $2m$ in both variables. (We can replace these operations by performing $O(r^2)$ times two-dimensional Fourier Transform at $m$ points in each dimension. We also need to perform some multiplications of vectors by diagonal matrices and $2r - 2$ subtractions of vector polynomials at the dominated computational cost.) At all $p$ stages of Newton's iteration we perform $O(pr^2)$ multiplications of bivariate polynomials of degrees at most $2m$ in both variables.

**Corollary 27.** *Let $A = I_m - \lambda B$ an $m \times m$ matrix, and assume that $Z_eA - AZ_f = GH^T$ for some $m \times r$ matrices $G$ and $H$. Then the powers $B^0, B, \ldots, B^m$ can be computed via Algorithm 21 ($k = m$) in parallel complexity*

$$O_A(\log(m)^2r^2, m^2r^2/\log(m)).$$

**The overall complexity of solving a Toeplitz-like linear system of equations.** It remains to estimate the complexity of performing Algorithm 22.

At Stage 1, we compute the trace of the matrix polynomial $A^{-1} \mod \lambda^m$ expressed via its short displacement generator by using Theorem 23. We compute (modulo $\lambda^m$) inner products of $m$ pairs of vector polynomials of dimension $m$ and then sum the $m$ computed values. Clearly, the overall computational cost of this operation is dominated by the estimated cost of performing Algorithm 21.

Next, recall that the trace of the matrix $B^k$ is the $k$th power sum of the eigenvalues of the matrix $B$, which are the roots of its characteristic polynomial $c_B$. At Stage 1, we produce these traces, equal to the power sums, and at Stage 2, we recover the coefficients of the characteristic polynomial from the power sums. We apply the solution algorithm for Problem 4.8 on pages 34–35 of [5] which amounts to performing $O(\log m)$ multiplications of polynomials of degrees at most $m$. Clearly the cost of performing this stage is even stronger dominated.

Stage 3 is reduced essentially to multiplication of $e$- and $f$-circulant matrices by $2m$ vectors. The cost of performing this stage is dominated by virtue of Theorem 24.

*Remark 28.* Appendix B of [30] extends the expression $B^{-1} = -\sum_{j=0}^{m}(c_i/c_0)B^{i-1}$ used at Stage 3 of Algorithm 22 to express the Moore–Penrose generalized inverse of a matrix through its characteristic polynomial. By employing this expression we can follow the paper [30] and readily extends the algorithms and the complexity estimates to the task of computing the least squares solution of a singular Toeplitz-like linear system of equations.

**Parallel complexity of the iteration in Definition 14.** Using the results of this section, we have the following corollary for the parallel complexity of modular inverse computation:

**Corollary 29.** *Let $q(T) \in \mathbb{K}(T)$ be degree $d$ and $p(T) \in \mathbb{K}(T)$ be degree at most $d-1$ that is relatively prime to $q(T)$. Then, we can compute $p^{-1}(T) \mod q(T)$ in parallel complexity $O_A(\log^2(d), d^2/\log(d))$.*

*Proof.* It follows from Corollary 27 and the previous subsection and from the fact that the Sylvester matrix of $p$ and $q$ has size $m \times m$ with $m \leq 2d-1$ and displacement rank $r \leq 2$.

Besides modular inverses, the computation of the polynomials in Definition 14 is dominated by the computation of the adjoint of the polynomial matrix $J_{\boldsymbol{F}}(\boldsymbol{v}(T))$ modulo $q(T)$. We can assume that all polynomials in the polynomial arithmetics involved, as well as our input polynomials in $\boldsymbol{F}$, have degree at most $2d$. Then, according to [5, page 311], the parallel complexity of division with remainder using degrees at most $2d$ and $d$ polynomials is $O_A(\log(d)\log^*(d), d/\log^*(d))$. Moreover, using [5, page 319], we can compute the adjoint (and the inverse) of an $n \times n$ scalar matrix in $O_A(\log^2(n), n^{\omega+1})$. Thus, the adjoint of $J_{\boldsymbol{F}}(\boldsymbol{v}(T))$ modulo $q(T)$ can be computed in

$$O_A(\log^2(n)\log(d)\log^*(d), n^{\omega+1}d/\log^*(d)).$$

Combining all the above we get the following proposition. Note that the most significant difference between its complexity bounds and the ones in Proposition 20 is the extra $d$ factor in the required number of processors.

**Proposition 30.** *Assume that we are given $\boldsymbol{F}$, $u$, $q(T)$ and $\mathbf{v}(T)$ satisfying Assumption 13. Assume further that the polynomials in $\boldsymbol{F}$ have degree at most $2d$. Then we can compute the polynomials $\bar{Q}(T), \bar{\boldsymbol{V}}(T) = (\bar{V}_1(T), \ldots, \bar{V}_n(T))$ of Definition 14 with the cost*

$$O_A(\log^2(n)\log^2(d)\log^*(d), n^{\omega+1}d^2/\log^*(d)).$$

**Simplified computation of cofactors from roots.** Given two coprime univariate polynomials $u = u(x)$ of a degree $d$ and $v = v(x)$ of a degree $m$ we seek their cofactors $s = s(x)$ of a degree at most $m-1$ and $t = t(x)$ of a degree at most $d-1$ such that $su+tv = 1$. This task amounts to the solution of a Sylvester linear system of equations, which we can compute by applying the algorithms of the previous subsections.

Next, we consider the case where we are given $d$ distinct roots $x_1, \ldots, x_d$ of the polynomial $u$. (We can proceed similarly where we are given $n$ distinct roots $y_1, \ldots, y_m$ of the polynomial $v$.) Then we can devise more efficient algorithms by applying the evaluation/interpolation techniques of [36] as follows.

**Algorithm 31** Computation of cofactors.

```
INPUT: two coprime univariate polynomials $u = u(x)$ of degree $d$ and $v = v(x)$
    of degree $m \leq d$ and $d$ distinct roots $y_1, \ldots, y_d$ of the polynomial $u$.
OUTPUT: two cofactors, $s = s(x)$ of a degree at most $m-1$ and $t = t(x)$ of a
    degree at most $d-1$ such that $su+tv = 1$.
COMPUTATIONS:
    1. Compute the values $t(y_i) = 1/v(y_i)$ for $i = 1, \ldots, d$.
    2. Interpolate the polynomial $t(x)$.
    3. Apply FFT to evaluate the polynomials $1-tv$ and $u$ and their ratio
       $(1-tv)/u$ at the $2^k$th roots of unity for $k = \lceil \log_2(d+m) \rceil$.
    4. Apply the inverse FFT to interpolate the polynomial $s = (1-tv)/u$.
```

At Stages 1 and 2 we apply the efficient known algorithms (see Sections 3.1, 3.3, and 3.13 of [31]) for the solution of both problems of multipoint evaluation and interpolation. These algorithms essentially amount to $O(\log(d))$ concurrent multiplications and divisions of univariate polynomials of degree at most $d$. The overall cost of performing these operations is substantially smaller than the cost of performing the algorithms of the previous subsections for the Sylvester Toeplitz-like linear systems of equations, and surely so is the cost of the application of FFT and inverse FFT at Stages 3 and 4 as well. Summarizing we perform the computations of the algorithm in $O(\log^2(d)\log^*(d), d/\log^*(d))$.

## References

1. J. Abbott, C. Fassino, and M.-L. Torrente. Stable border bases for ideals of points. *J. Symbolic Comput.*, 43(12):883–894, 2008.
2. T.A. Akoglu, J.D. Hauenstein, and A. Szanto. Certifying solutions to overdetermined and singular polynomial systems over Q. manuscript, 2013.
3. M. Avendaño, T. Krick, and A. Pacetti. Newton-Hensel interpolation lifting. *Found. Comput. Math.*, 6(1):81–120, 2006.
4. D.J. Bates, J.D. Hauenstein, A.J. Sommese, and C.W. Wampler. Bertini: software for numerical algebraic geometry. Available at bertini.nd.edu.
5. D. Bini and V. Y. Pan. *Polynomial and matrix computations. Vol. 1.* Progress in Theoretical Computer Science. Birkhäuser Boston Inc., Boston, MA, 1994. Fundamental algorithms.
6. L. Blum, F. Cucker, M. Shub, and S. Smale. *Complexity and real computation.* Springer-Verlag, New York, 1998. With a foreword by Richard M. Karp.
7. D. Castro, L. M. Pardo, K. Hägele, and J. E. Morais. Kronecker's and Newton's approaches to solving: a first comparison. *J. Complexity*, 17(1):212–303, 2001.
8. A. L. Chistov. An algorithm of polynomial complexity for factoring polynomials, and determination of the components of a variety in a subexponential time. *Zap. Nauchn. Sem. Leningrad. Otdel. Mat. Inst. Steklov. (LOMI)*, 137:124–188, 1984. Theory of the complexity of computations, II.
9. C. Fassino. Almost vanishing polynomials for sets of limited precision points. *J. Symbolic Comput.*, 45(1):19–37, 2010.
10. C. Fassino and M.-L. Torrente. Simple varieties for limited precision points. *Theoret. Comput. Sci.*, 479:174–186, 2013.
11. H. R. P. Ferguson, D. H. Bailey, and S. Arno. Analysis of PSLQ, an integer relation finding algorithm. *Math. Comp.*, 68(225):351–369, 1999.
12. P. Giorgi, C. Jeannerod, and G. Villard. On the complexity of polynomial matrix computations. In *Proceedings of the 2003 International Symposium on Symbolic and Algebraic Computation*, pages 135–142. ACM Press, 2003.
13. M. Giusti, J. Heintz, K. Hägele, J. E. Morais, L. M. Pardo, and J. L. Montaña. Lower bounds for Diophantine approximations. *J. Pure Appl. Algebra*, 117/118:277–317, 1997. Algorithms for algebra (Eindhoven, 1996).
14. M. Giusti, J. Heintz, J. E. Morais, J. Morgenstern, and L. M. Pardo. Straight-line programs in geometric elimination theory. *J. Pure Appl. Algebra*, 124(1-3):101–146, 1998.
15. M. Giusti, G. Lecerf, and B. Salvy. A Gröbner free alternative for polynomial system solving. *J. Complexity*, 17(1):154–211, 2001.
16. D. Y. Grigor′ev. Factoring polynomials over a finite field and solution of systems of algebraic equations. *Zap. Nauchn. Sem. Leningrad. Otdel. Mat. Inst. Steklov. (LOMI)*, 137:20–79, 1984. Theory of the complexity of computations, II.
17. J.D. Hauenstein. Numerically computing real points on algebraic sets. *Acta Appl. Math.*, 125(1), 105–119, 2013.
18. J. D. Hauenstein and F. Sottile. Algorithm 921: alphaCertified: certifying solutions to polynomial systems. *ACM Trans. Math. Software*, 38(4):Art. ID 28, 20, 2012.
19. J. Heintz, T. Krick, S. Puddu, J. Sabia, and A. Waissbein. Deformation techniques for efficient polynomial equation solving. *J. Complexity*, 16(1):70–109, 2000.
20. D. Heldt, M. Kreuzer, S. Pokutta, and H. Poulisse. Approximate computation of zero-dimensional polynomial ideals. *J. Symbolic Comput.*, 44(11):1566–1591, 2009.

21. G. Jeronimo, T. Krick, J. Sabia, and M. Sombra. The computational complexity of the Chow form. *Found. Comput. Math.*, 4(1):41–117, 2004.
22. G. Jeronimo and D. Perrucci. On the minimum of a positive polynomial over the standard simplex. *J. Symbolic Comput.*, 45(4):434–442, 2010.
23. E. Kaltofen. Polynomial-time reductions from multivariate to bi- and univariate integral polynomial factorization. *SIAM J. Comput.*, 14(2):469–489, 1985.
24. E. Kaltofen. Sparse Hensel lifting. In *EUROCAL '85, Vol. 2 (Linz, 1985)*, volume 204 of *Lecture Notes in Comput. Sci.*, pages 4–17. Springer, Berlin, 1985.
25. R. Kannan, A. K. Lenstra, and L. Lovász. Polynomial factorization and nonrandomness of bits of algebraic and some transcendental numbers. *Math. Comp.*, 50(181):235–250, 1988.
26. R. Karp and V. Ramachandran. Parallel algorithms for shared-memory machines. *Handbook of theoretical computer science*, Vol. A, 869941, Elsevier, Amsterdam, 1990.
27. A. K. Lenstra, H. W. Lenstra, Jr., and L. Lovász. Factoring polynomials with rational coefficients. *Math. Ann.*, 261(4):515–534, 1982.
28. D. Lichtblau. Exact computation using approximate Gröbner bases. Available in the Wolfram electronic library, 2010.
29. B. Mourrain and P. Trébuchet. Stable normal forms for polynomial system solving. *Theoret. Comput. Sci.*, 409(2):229–240, 2008.
30. V.Y. Pan. Parametrization of Newton's iteration for computations with structured matrices and applications. *Comput. Math. Appl.*, 24(3):61–75, 1992.
31. V. Y. Pan, *Structured Matrices and Polynomials: Unified Superfast Algorithms*, Birkhäuser/Springer, Boston/New York, 2001.
32. F. Rouillier. Solving zero-dimensional systems through the rational univariate representation. *Journal of Applicable Algebra in Engineering, Communication and Computing*, 9(5):433–461, 1999.
33. K. Shirayanagi. An algorithm to compute floating point Groebner bases. In *Mathematical computation with Maple V: ideas and applications (Ann Arbor, MI, 1993)*, pages 95–106. Birkhäuser Boston, Boston, MA, 1993.
34. K. Shirayanagi. Floating point Gröbner bases. *Math. Comput. Simulation*, 42(4-6):509–528, 1996. Symbolic computation, new trends and developments (Lille, 1993).
35. H. J. Stetter. *Numerical polynomial algebra.* Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2004.
36. A. L. Toom. The Complexity of a Scheme of Functional Elements Realizing the Multiplication of Integers, *Soviet Mathematics Doklady*, **3**, 714–716, 1963.
37. C. Traverso and A. Zanoni. Numerical stability and stabilization of Groebner basis computation. In *Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation*, pages 262–269 (electronic). ACM, New York, 2002.
38. W. Trinks. On improving approximate results of Buchberger's algorithm by newton's method. In B. Caviness, editor, *EUROCAL '85*, volume 204 of *Lecture Notes in Computer Science*, pages 608–612. Springer Berlin Heidelberg, 1985.
39. C.W. Wampler, J.D. Hauenstein, and A.J. Sommese. Mechanism mobility and a local dimension test. *Mech. Mach. Theory*, 46(9), 1193–1206, 2011.
40. C.W. Wampler, B. Larson, and A. Edrman. A new mobility formula for spatial mechanisms. In *Proc. DETC/Mechanisms & Robotics Conf., Sept. 4–7, Las Vegas, NV (CDROM)*, 2007.
41. F. Winkler. A $p$-adic approach to the computation of Gröbner bases. *J. Symbolic Comput.*, 6(2-3):287–304, 1988. Computational aspects of commutative algebra.
42. H. Zassenhaus. On Hensel factorization. I. *J. Number Theory*, 1:291–311, 1969.