Geometric Aspects of Sparse Multivariate Rational Function Interpolation with Application to Solving Parametric Linear Systems

by

Archit Srivastava

M.S., University of Nebraska, Lincoln 2022
B.Tech., Dr. Abul Kalam Azad Technical University, 2017

Project Submitted in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

in the
Department of Mathematics
Faculty of Science

SIMON FRASER UNIVERSITY
Spring 2026

Declaration of Committee

Name:	**Archit Srivastava**
Degree:	**Master of Science**
Project title:	**Geometric Aspects of Sparse Multivariate Rational Function Interpolation with Application to Solving Parametric Linear Systems**
Committee:	**Michael Monagan** Supervisor Professor, Mathematics
	Ladislav Stacho Reader Professor, Mathematics

Abstract

In this work, we implement the Kaltofen and Yang sparse multivariate rational function interpolation algorithm, incorporating Monagans maximal quotient rational function reconstruction and the Ben-OrTiwari method for multivariate polynomial interpolation. We further investigate the geometric structure underlying the Kaltofen Yang algorithm, a sparse multivariate rational function interpolation algorithm, and apply it to solve parametric systems of linear equations.

Keywords: Kaltofen Yang sparse multivariate rational function interpolation, geometric interpretation; Simon Fraser University;

Dedication

To Maa, Papa, Futbulli, Mamu and the loving memory of my grandparents- Mummy and Nati.

Acknowledgements

I wish to express my deepest gratitude to my supervisor, Dr. Monagan. This work would not have been possible without his tremendous guidance, unwavering patience, and constructive feedback.

I would also like to thank Professor Katrina Honigs for her kindness and monumental patience during my coursework in MATH 741 and MATH 819. I also extend my thanks to Graham, Chiki, and Pijush for listening to my ideas and answering my many questions in algebraic geometry.

I am also grateful to the amazing friends I have had the privilege of meeting during my time in the SFU Mathematics Department. Thank you for making my time at SFU memorable, Mantej, Stephan, Carl, Aniket, Ming, Alex, Leo, Rebekah, Alicia, Deniz, Shannon, Negarin, Sophie, Hyukh, Jingzhou, Magdalena, Marlene, and Diya.

I would also like to acknowledge Anthony and the wonderful EB community for their support.

Last but not least, I thank my found family-Sheelu, Pinkuu, and Pundu-for their enduring support and for standing by me through thick and thin.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Let $\mathbb{F}$ be a base field. Consider a multivariate rational function h in $x_1, \ldots, x_n$ over $\mathbb{F}(x_1, \ldots, x_n)$ that we want to interpolate. Let,

$$h(x_1, \ldots, x_n) = \frac{f(x_1, \ldots, x_n)}{g(x_1, \ldots, x_n)} \text{ where } f(x_1, \ldots, x_n),\ g(x_1, \ldots, x_n) \in \mathbb{F}[x_1, \ldots, x_n].$$

The Kaltofen Yang sparse multivariate rational function interpolation algorithm [14] interpolates $h(x_1, \ldots, x_n)$ by interpolating $\mu f(x_1, \ldots, x_n)$ and $\mu g(x_1, \ldots, x_n)$ for some fixed $\mu \in \mathbb{F}$. So, the Kaltofen Yang algorithm converts the problem of interpolation of rational functions into polynomial interpolation in $\mathbb{F}[x_1, \ldots, x_n]$. In this work, we use the Ben-Or Tiwari sparse multivariate polynomial interpolation [2] to interpolate $\mu f(x_1, \ldots, x_n)$ and $\mu g(x_1, \ldots, x_n)$ separately.

Kaltofen and Yang introduce an ingenious ring morphism ϕ_1 defined using points $\beta = [1, \beta_2, \ldots, \beta_n]^T,\ \sigma = [\sigma_1, \sigma_2, \ldots, \sigma_n]^T \in \mathbb{F}^n$. Under this ring morphism, the image of h is a univariate rational function $T(x) = \dfrac{\hat{f}(x)}{\hat{g}(x)} \in \mathbb{F}(x)$, such that

$$T(x) = \phi_1(h) = \frac{\phi_1(f)}{\phi_1(g)} = \frac{f(x,\, \beta_2(x-\sigma_1)+\sigma_2,\, \ldots,\, \beta_n(x-\sigma_1)+\sigma_n,\, \sigma_1)}{g(x,\, \beta_2(x-\sigma_1)+\sigma_2,\, \ldots,\, \beta_n(x-\sigma_1)+\sigma_n,\, \sigma_1)} = \frac{\hat{f}(x)}{\hat{g}(x)}$$

Here $\hat{f}(x),\ \hat{g}(x) \in \mathbb{F}[x]$, a Euclidean domain. Monagan's maximal quotient rational function reconstruction algorithm [17] interpolates $T(x)$ and returns $\mu\hat{f}$(x) and $\mu\hat{g}(x)$, for some fixed $\mu \in \mathbb{F}$.
We then use Ben-Or Tiwari multivariate polynomial interpolation to recover μf and μg, separately from $\mu\hat{f}(\sigma_1)$ and $\mu\hat{g}(\sigma_1)$ for different values of $\sigma \in \mathbb{F}^n$.

We also explore the geometric properties of the ring morphism ϕ_1 characterized by the points $\beta, \sigma \in \mathbb{F}^n$ and show that it defines a parameterized affine line $\mathcal{L}$ passing through the point σ in the direction of β, in the ambient space of the rational function $h(x_1, \ldots, x_n)$

and we interpret the working of the Kaltofen Yang algorithm geometrically.

We then apply the Kaltofen Yang algorithm to the problem of solving a $n \times n$ linear system $Ax = b$ in m parameters $y_1, \ldots, y_m$. In such a system, the entries a_{ij} where $1 \leq i, j \leq n$ of the $n \times n$ matrix A are polynomials in $\mathbb{F}[y_1, \ldots, y_m]$. The solution of such a system is a n dimensional vector of multivariate rational functions in $\mathbb{F}(y_1, \ldots, y_m)$, i.e.

$$x_k = \frac{f_k(y_1, \ldots, y_m)}{g_k(y_1, \ldots, y_m)}, \text{ for } 1 \leq k \leq n.$$

We first evaluate the polynomial entries in the matrix A at distinct points α_i chosen randomly from the parameter space $\mathbb{F}^m$, giving us a matrix of constants $A(\alpha_i) \in \mathbb{F}^{n \times n}$ at α_i. We solve this system to obtain the numerical solution at each evaluation point α_i. We then use the Kaltofen Yang multivariate rational function interpolation algorithm to recover each component x_k separately.
The solutions of the linear parameterized system $Ax = b$ characterize the co-ordinate functions of a rational map Φ from $\mathbb{F}^m$ to $\mathbb{F}^n$. Using the Kaltofen Yang algorithm to find the solutions of a parametric linear system, we recover the rational map Φ. Thus, we can generalize the Kaltofen Yang algorithm from interpolation of multivariate rational functions to recovery of rational maps, characterized by the solutions of a linear parametric system.

1.1 Example: Multivariate rational function

Consider the rational function h in x_1 and x_2 over $\mathbb{Q}$

$$h(x_1, x_2) = \frac{f(x_1, x_2)}{g(x_1, x_2)} \text{ where } f(x_1, x_2) = x_1 - x_2, \quad g(x_1, x_2) = \frac{4}{5}x_1^3 + \frac{2}{5}x_1x_2 - \frac{3}{4}x_2^2 + 1$$

The Kaltofen Yang [14] sparse multivariate rational function interpolation algorithm converts the problem of interpolation of $h(x_1, x_2) \in \mathbb{Q}(x_1, x_2)$ into polynomial interpolation by interpolating $\mu f(x_1, x_2)$ and $\mu g(x_1, x_2) \in \mathbb{Q}[x_1, x_2]$ for some fixed $\mu \in \mathbb{Q}$. Using the ring morphism ϕ_1, defined using points $\beta = [1, \beta_2]^2, \sigma = [\sigma_1, \sigma_2]^T \in \mathbb{F}^2$ we find the univariate image $T(x) = \frac{\hat{f}(x)}{\hat{g}(x)} \in \mathbb{Q}(x)$ of the rational function h such that $h(\sigma_1, \sigma_2) = T(\sigma_1)$, we then recover $\mu f(x_1, x_2)$, $\mu g(x_1, x_2) \in \mathbb{Q}[x_1, x_2]$ using the Ben-Or Tiwari algorithm.
Implementing Kaltofen Yang with the points chosen from $\mathbb{Q}$ results in large rational numbers throughout the algorithm due to intermediate expression swell. To avoid this, we modify it to work modulo a sequence of primes and employ Chinese remaindering and Wang's rational number reconstruction [20] to recover the rational number coefficients appearing in $h(x_1, x_2)$.

1.1.1 Example: Parametric linear system

Consider the following 2×2 system of linear equations with parameters y_1 and y_2 that we wish to solve.

$$y_1 x_1 + y_1 x_2 = 1 \tag{1.1}$$

$$y_1 y_2 x_1 - x_2 = 1 \tag{1.2}$$

Each equation of this system is a polynomial in $\mathbb{Z}[x_1, x_2, y_1, y_2]$. Re-writing the system as $A\mathbf{x} = b$ gives us,

$$A := \begin{bmatrix} y_1 & y_1 \\ y_1 y_2 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

The coefficient matrix $A := \begin{bmatrix} y_1 & y_1 \\ y_1 y_2 & -1 \end{bmatrix}$ contains entries in $\mathbb{Z}[y_1, y_2]$. The augmented matrix for the system is given by,

$$\left[\begin{array}{cc|c} y_1 & y_1 & 1 \\ y_1 y_2 & -1 & 1 \end{array}\right]$$

The solution vector $\mathbf{x} \in \mathbb{Z}(y_1, y_2)^2$ is a vector of multivariate rational functions with components $x_1, x_2 \in \mathbb{Z}(y_1, y_2)$.

$$x_1 = \frac{y_1 + 1}{y_1^2 y_2 + y_1}, \quad x_2 = \frac{y_2 - 1}{y_1 y_2 + 1}. \tag{1.3}$$

The solution of this system characterizes the coordinate functions of a rational map $\Phi : \mathbb{Z}^2 \longrightarrow \mathbb{Z}^2$. Thus, finding the solutions to this system recovers Φ.

1.2 A brief overview of computer arithmetic

There are multiple computational paradigms for performing arithmetic on modern computers. The following are the most widely used

1. Fixed point arithmetic.
2. Floating point arithmetic, defined by the IEEE 754 standard.
3. Multi-precision arithmetic.

Floating-point arithmetic uses fixed operand size-32 bits for single precision and 64 bits for double precision. By employing rounding and truncation, it manages the numerical range effectively to prevent overflow. Because these operand sizes are standardized, operations such as addition and multiplication can be hardwired directly into CPUs and GPUs via dedicated Floating-Point Units (FPUs). This hardware integration makes floating-point arithmetic exceptionally fast, though it is lossy due to precision limits.
Computer Algebra Systems use the multi-precision, also known as arbitrary-precision, paradigm

for exact or symbolic computation. Unlike standard floating-point arithmetic, which is constrained by fixed operand sizes, multi-precision arithmetic accommodates operands of varying sizes. Multi-precision arithmetic is not natively supported by general-purpose hardware and is implemented through software libraries that are often optimized at the assembly level for different processors. Although there is no universal standard like IEEE 754 for multi-precision arithmetic, GNU's GMP (GNU Multiple Precision Arithmetic Library) and MPFR (GNU Multiple Precision Floating-Point Reliable Library) are the most widely used multi-precision libraries and act as functional standards. These libraries are the bedrock for Computer Algebra Systems (Maple), cryptography libraries (OpenSSL, libgcrypt), and computational geometry libraries (CGAL).

1.2.1 Intermediate expression swell

Because multi-precision arithmetic imposes no upper bound on operand size, calculations can suffer from a dramatic blow-up in the size of intermediate results. This phenomenon, known as expression swell, significantly slows down computation as the processor struggles to manage increasingly massive data structures. We therefore pick a prime p and use modular arithmetic in a finite field $\mathbb{Z}_p$.

1.2.2 Modular Arithmetic.

For a prime p, the elements of the field $\mathbb{Z}_p$ are less than p, this bounds the sizes of the operands from above, preventing the intermediate expression swell. After doing all the arithmetic in $\mathbb{Z}_p$, we can use the Chinese remainder theorem [19] or Wang's rational reconstruction algorithm [20] to recover the final result in $\mathbb{Z}$ or $\mathbb{Q}$ respectively.

Modular reconstruction

The field $\mathbb{Z}_p$ consists of equivalence classes of residues modulo p. $\mathbb{Z}$ and $\mathbb{Q}$ are infinite therefore, the mapping into $\mathbb{Z}_p$ is many-to-one. To ensure the modular reconstruction is successful, an important question is how many prime numbers to use. Let $p_1, \ldots, p_w$ be w distinct primes such that $M = \prod_{i=1}^{w} p_i$. The Chinese Remainder Theorem (CRT) reconstructs the solution uniquely if it lies within the symmetric interval [19],

$$\left(-\frac{M}{2}, \frac{M}{2}\right]$$

If the true result falls outside this window, it will be reconstructed as an incorrect value. Thus, unambiguous recovery requires a magnitude bound on the result to guarantee that the chosen modulus M covers the necessary range.

Rational reconstruction bound

Let $p_1, \ldots, p_w$ be w distinct primes such that $M = \prod_{i=1}^{w} p_i$. Suppose that we want to recover a reduced rational number

$$\frac{n}{d} \in \mathbb{Q}, \text{ such that } \gcd(n, d) = 1, \quad d > 0,$$

such that $\gcd(p_i, d) = 1$, where $1 \leq i \leq w$. Then in order to recover $n, d \bmod M$ from $u = \frac{n}{d} \bmod M$, the following relation [20] should hold,

$$M > 2(|n| \cdot |d|). \tag{1.4}$$

Where $|n|, |d|$ are the sizes of the numerator and denominator.

Hadamard Bound

For a matrix A, the Hadamard bound $H(A)$ [19] gives an upper bound for the absolute value of the determinant of A.

$$\det(A) \leq \prod_{i=1}^{n} \sqrt{\sum_{j=1}^{n} a_{i,j}^2} = \prod_{j=1}^{n} \|r_j\|_2 = H(A)$$

where r_j are the rows of the matrix A. To recover the determinant in the symmetric range $(-\frac{M}{2}, \frac{M}{2}]$ without ambiguity, $M > 2H(A)$.

Rational reconstruction bound for linear systems

For a linear system $Ax = b$ where $A \in \mathbb{Z}^{n \times n}, b \in \mathbb{Z}^n$, the solution over $\mathbb{Q}$ is $x = A^{-1}b$. Even if entries of A and b are small, A^{-1} can have large rational entries. Recall Cramer's rule, $x_i = \dfrac{\det(A_i)}{\det(A)}$, where A_i is the matrix A with its i^{th} column replaced by b. To ensure valid rational reconstruction, the modulus M is chosen by combining the rational reconstruction condition with Hadamard's bound,

$$M > 2H(A) \cdot \max_{1 \leq i \leq n} H(A_i)$$

In practice, we use 32-bit or 64-bit machine primes.

1.3 Black boxes

In computer algebra, the "black box model" refers primarily to an implicit computational model for an arbitrary element over some algebraic object [13]. A black box is implemented as a procedure in a Computer Algebra System that accepts appropriate input from the user to pick a particular element from the underlying algebraic object, perform the appropriate

computations, and return the output, if one exists or return a fail message. Consider the following examples over some commutative ring R and field $\mathbb{F}$ that can be represented as a black box which returns,

1. the value of $f(\alpha_1, \ldots, \alpha_n) \in R$ or $\mathbb{F}$ for some polynomial $f(x_1, \ldots, x_n) \in R[x_1, \ldots, x_n]$ or $\mathbb{F}[x_1, \ldots, x_n]$ at some point $\alpha \in R^n$ or $\mathbb{F}^n$.

2. the value of $\frac{f(\alpha_1,\ldots,\alpha_n)}{g(\alpha_1,\ldots,\alpha_n)} \in \mathbb{F}$ for some rational function $\frac{f(x_1,\ldots,x_n)}{g(x_1,\ldots,x_n)}$ over a field of rational functions $\mathbb{F}(x_1, \ldots, x_n)$.

3. the determinant $\det(A) \in R[y_1, \ldots, y_m]$ of some matrix A in the ring of $n \times n$ matrices $M_n(R[y_1, \ldots, y_m])$.

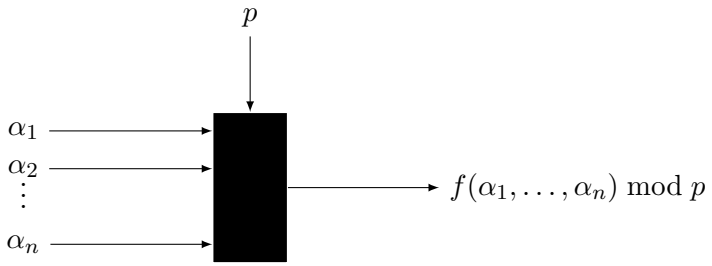
4. the solution vector $x \in \mathbb{F}(y_1, \ldots, y_m)^n$ for some parametric system of linear equations $Ax = b$, where $A \in M_n(\mathbb{F}[y_1, \ldots, y_m])$ and $b \in \mathbb{F}^n$.

The black box $\mathcal{B}$ of an algebraic object hides its internal structure and complexity and emulates certain underlying mathematical properties, like the linearity property in the case of polynomial black boxes. Probing the black box of an object produces its image under the evaluation homomorphism.

1.4 Polynomial black boxes and evaluation homomorphisms

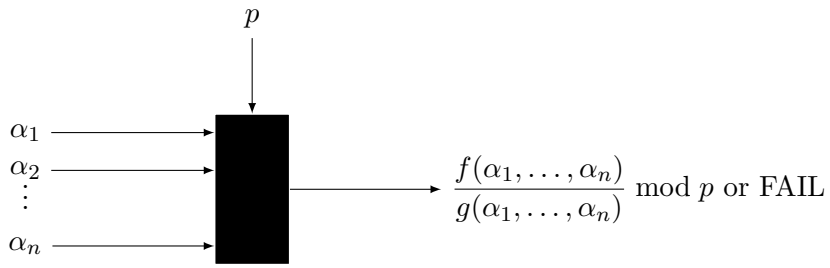
1.4.1 Modular black box for a multivariate polynomial

Definition 1.4.1. *Let f be a polynomial in n variables $x_1, \ldots, x_n$ over a field $\mathbb{F}$ ie. $f \in \mathbb{F}[x_1, \ldots, x_n]$. A modular black box $\mathcal{B}$ for f is a program that on input $\alpha \in \mathbb{F}^n$ and a prime p computes, $f(\alpha) \mod p$ ie. $\mathcal{B} : (\mathbb{F}^n, p) \longrightarrow \mathbb{F}_p$.*



1.4.2 Modular black boxes for rational functions

Definition 1.4.2. *Let $\frac{f}{g}$ be a rational polynomial in n variables $x_1, \ldots, x_n$ over a field of rational functions $\mathbb{F}$ ie. $\frac{f}{g} \in \mathbb{F}(x_1, \ldots, x_n)$. A modular black box $\mathcal{B}$ for $\frac{f}{g}$ is a program that on input $\alpha \in \mathbb{F}^n$ and a prime p computes, $\frac{f(\alpha)}{g(\alpha)} \mod p$ ie. $\mathcal{B} : (\mathbb{F}^n, p) \longrightarrow \mathbb{F}_p$. If $g(\alpha) = 0$ it outputs FAIL.*



Theorem 1.4.1. *Evaluating a modular black for a multivariate polynomial $f \in \mathbb{F}[x_1, \ldots, x_n]$ at t distinct points will produce a vector in $\mathbb{F}_p^t$.*

Proof. Let $\mathcal{B}$ be a modular black box, let $\alpha_i = \begin{bmatrix} \alpha_{i1} \\ \vdots \\ \alpha_{in} \end{bmatrix} \in \mathbb{F}^n$, where $1 \leq i \leq t$ be the points at which we evaluate $\mathcal{B}$ for some prime p. We know that each point α_i corresponds to the maximal ideal $\mathcal{I}_i = (x_1 - \alpha_{i1}, \ldots, x_n - \alpha_{in}) \subset \mathbb{F}[x_1, \ldots, x_n]$.
From the Chinese remainder theorem [7] we get that the map

$$\phi : \quad \mathbb{F}[x_1, \ldots, x_n] \quad \longrightarrow \quad \prod_{i=1}^{t} \mathbb{F}[x_1, \ldots, x_n]/\mathcal{I}_i$$

is a ring homomorphism with kernel $\bigcap_{i=1}^{t} \mathcal{I}_i$. If the ideals are distinct then $\bigcap_{i=1}^{t} \mathcal{I}_i = \prod_{i=1}^{t} \mathcal{I}_i$ and the map ϕ is surjective. Moreover,

$$\implies \prod_{i=1}^{t} \mathbb{F}[x_1, \ldots, x_n]/\mathcal{I}_i \cong \mathbb{F}[x_1, \ldots, x_n]/\prod_{i=1}^{t} \mathcal{I}_i \cong \mathbb{F}^t. \tag{1.5}$$

Evaluating the modular black box $\mathcal{B}$ at multiple points give us $\prod_{i=1}^{t} \mathbb{F}[x_1, \ldots, x_n]/\mathcal{I}_i \cong \mathbb{F}^t$. □

1.5 Randomized algorithms and early termination

1.5.1 Randomized algorithms

While the use of modular black boxes significantly reduces arithmetic costs, it introduces a fundamental challenge: the shift from a deterministic paradigm to a probabilistic one. The primary risk involves encountering an "unlucky prime"a modulus that divides a critical value, such as a determinant or a leading coefficient. Consequently, this value vanishes in the modular field, potentially causing the algorithm to incorrectly identify a matrix as singular or a polynomial as zero.
To ensure correctness, systems typically employ Las Vegas algorithms. These methods detect such algebraic failures and restart with new primes, or strictly enforce a magnitude bound (such as the Hadamard Bound). Once the product of the chosen primes exceeds

this theoretical bound, the result can be unambiguously reconstructed using the Chinese Remainder Theorem.

1.5.2 Las Vegas algorithms

A Las Vegas algorithm is a randomized algorithm that always produces the correct result, but its running time is a random variable.
Consider the following example of a 2×2 matrix determinant computation,

$$A = \begin{bmatrix} 50 & 12 \\ 30 & 10 \end{bmatrix}$$

The Hadamard bound $H(A) = \prod_{i=1}^{2} \sqrt{\sum_{j=1}^{2} a_{i,j}^2} \sim 911$. To reconstruct the result uniquely within the symmetric range $(-\frac{M}{2}, \frac{M}{2}]$, the modulus M must satisfy $M > 2H(A) \approx 1820$. Let $p_1 = 7$, then

$$A \mod 7 = \begin{bmatrix} 1 & 5 \\ 2 & 3 \end{bmatrix}, \qquad \det(A_7) = 3 - 10 = -7 \equiv 0 \mod 7.$$

Since $\det(A_7) = 0$, the Las Vegas algorithm will select another prime to verify is whether A is singular or 7 is an unlucky prime.

$$A_{11} = \begin{bmatrix} 6 & 1 \\ 8 & 10 \end{bmatrix}, \qquad \det(A_{11}) = 8.$$

Since $\det(A_{11}) \neq 0$, we have verified that A is non singular. We can pick more primes as specified by the Hadamard bound. $11 \cdot 13 \cdot 17 = 2431 > 1820$.

$$A_{13} = \begin{bmatrix} 11 & 12 \\ 4 & 10 \end{bmatrix}, \qquad \det(A_{13}) = 10.$$

$$A_{17} = \begin{bmatrix} 16 & 12 \\ 13 & 10 \end{bmatrix}, \qquad \det(A_{17}) = 4.$$

The Chinese remainder theorem recovers $\det(A) = 140$ from, $x \equiv 8 \pmod{11}$, $x \equiv 10 \pmod{13}$, $x \equiv 4 \pmod{17}$.

1.5.3 Failure probability

If we have a modular black box $\mathcal{B}$ for a non-zero polynomial over a field $\mathbb{F}$, that takes a point $\alpha \in \mathbb{F}^n$ and a prime p as input and returns $f(\alpha) \mod p$ as output, then the probability that $\mathcal{B}(\alpha, p) = 0$ is given by the Schwartz-Zipple lemma [19].

Lemma 1.5.1 (Schwartz-Zippel lemma)**.** *Let $f(x_1, \ldots, x_n) \in \mathbb{F}[x_1, \ldots, x_n]$ be such that $f \neq 0$ and $S \subset \mathbb{F}$ be large and finite. If we randomly sample α from S^n then*

$$Pr[f(\alpha) = 0] \leq \frac{\deg(f)}{|S|}.$$

Let $h(x_1, \ldots, x_n) = \frac{f(x_1,\ldots,x_n)}{g(x_1,\ldots,x_n)} \in \mathbb{F}(x_1, \ldots, x_n)$ be such that $g \neq 0$ and let $\mathcal{B} : (\mathbb{F}^n, p) \longrightarrow \mathbb{F}_p$ be the modular black box for h. Then rational function interpolation algorithm would return FAIL, for any point $\alpha \in \mathbb{F}^n$ such that $g(\alpha) = 0$. We can use Schwartz-Zippel lemma to find the failure probability for the algorithm. Since $g(x_1, \ldots, x_n) \in \mathbb{F}[x_1, \ldots, x_n]$,

$$Pr[FAIL] = Pr[g(\alpha) = 0] \leq \frac{\deg(g)}{|S|}$$

If we choose a large prime p, and let S be equal to $\mathbb{F}_p$, we can make the failure probability as low as we desire, thus making sure that the algorithm gives the correct output with high probability (whp). We can use another point $\beta \neq \alpha$ to reduce the probability of failure,

$$Pr[g(\alpha) = 0 \wedge g(\beta) = 0] \leq \frac{\deg(g)^2}{|S| \cdot |S - 1|}$$

1.5.4 Early termination

While using black boxes for polynomial interpolation, we loose information about the polynomials like their degree and the number of terms they contain, while we can find the degree of the polynomial using interpolation, we cannot bound the number of terms of the polynomial. This makes it challenging to come up with a bound that acts as a termination condition beforehand for an iterative algorithm using the black box.
Numerical algorithms use the notion of convergence of error as a termination condition. We need something analogous for algorithms that use black boxes for polynomials.
Early termination is a probabilistic technique in black box sparse polynomial interpolation that allows algorithms to adaptively detect and terminate once the polynomial is interpolated, without requiring any information beforehand regarding the degree and the number of monomials of the polynomial. This often dramatically reduces black box probes to the black box. Suppose that we randomly pick $\alpha_0, \alpha_1, \alpha_2 \ldots$, from S and compute $y_i = f(\alpha_i)$. Let $\hat{f}_i(x)$ be the interpolation polynomial for the points (α_k, y_k) for $0 \leq k \leq i$, when $\hat{f}_l(x) = \hat{f}_{l+1}(x)$, $\hat{f}_l = f(x)$ with high probability. Let $f(x) \in \mathbb{F}_p[x]$ and $S \subset \mathbb{F}_p$ be large and finite. We randomly sample points s_i from S for polynomial interpolation. Once the correct interpolating polynomial $\hat{f}$ has been reconstructed, future probes must satisfy $f(\alpha) = \hat{f}(\alpha)$. Thus, we can terminate the interpolation algorithm once the interpolating polynomial stabilizes i.e. $\hat{f}_{l+1} = \hat{f}_l$.

1.6 Sparse polynomials

Polynomials can be classified into two categories based on the density of their non-zero terms relative to the maximum possible number of terms. Suppose $f \in \mathbb{F}[x_1, \ldots, x_n]$ with $\deg(f, x_i) = d_i$, and total $\deg f = d$. Then the maximum number of terms $M = \prod_{i=1}^{n}(d_i + 1) \in O(d^n)$. Sparse multivariate polynomial interpolation algorithms like Ben-Or, Tiwari, and Zippel's algorithm have a complexity that scales with the number of non-zero terms (t) rather than the maximum possible terms (M).

1.6.1 Sparse polynomial definition and example

Definition 1.1. Let f be a non-zero polynomial in $x_1, x_2, \ldots, x_n$. Let t denote the number of non-zero terms in f such that $t \geq 1$ and let $d = \deg(f)$ be the total degree of f. The maximum number of possible terms in f is

$$M = \binom{n+d}{d}.$$

We say that f is *sparse* if $t < \sqrt{M}$.

Example 1.2. Let $f = x_1 x_2^3 x_5 + 2x_1 x_3 x_4$. Here $t = 2$, $d = 5$, $n = 5$, and

$$M = \binom{5+5}{5} = \binom{10}{5} = 252.$$

Therefore, f is a sparse polynomial since

$$t = 2 < \sqrt{M} = \sqrt{252} \approx 15.9.$$

Definition 1.3. A sparse polynomial f is normally represented as

$$f = \sum_{k=1}^{t} c_k \, x_1^{e_{k,1}} x_2^{e_{k,2}} \cdots x_n^{e_{k,n}}, \qquad \text{where } c_k \neq 0.$$

Definition 1.4. A multivariate rational function f/g with $\gcd(f, g) = 1$ is said to be*sparse* if both polynomials f and g are sparse.

1.6.2 Sparse multivariate polynomial interpolation à la Ben-Or Tiwari

Let $f = \sum_{i=1}^{t} a_i \mathcal{M}_i$ where $a_i \in \mathbb{Z}$, $\mathcal{M}_i = \prod_{j=1}^{n} x_j^{e_i j}$. Ben-Or Tiwari [2] is a two-step multivariate interpolation algorithm that interpolates all the variables of the polynomial simultaneously as opposed to Zippel's multivariate interpolation algorithm, which interpolates the variables one at a time.

For efficiency, we will use Ben-Or Tiwari algorithm to interpolate $f(x_1, \ldots, x_n) \in \mathbb{Z}[x_1, \ldots, x_n]$

modulo a prime p. We use the 31 bit prime $p = 2^{31} - 1$ for our computations.
Let $\mathcal{B}$ be the modular black box for $f(x_1, \ldots, x_n) \in \mathbb{Z}[x_1, \ldots, x_n]$. We evaluate the black box $\mathcal{B}$ at the point $\sigma_i = [2^i, 3^i, 5^i, \ldots, \mathfrak{p}_n^i]^T \in \mathbb{Z}_p^n$, where $\mathfrak{p}_n$ is the n^{th} prime. This gives us

$$v_i = f(2^i, 3^i, 5^i, \ldots, \mathfrak{p}_n^i) \mod p, \text{ for } 0 \leq i < 2t,$$

where t is the number of non zero monomials of f.
We will show in chapter 3 that $\{f(2^i, 3^i, 5^i, \ldots, \mathfrak{p}_n^i)\}_{0 \leq i \leq 2t}$ forms a linear recursive sequence of order t. In the first step of the Ben-Or Tiwari algorithm, we use the Berkekamp Massey Euclidean algorithm (BMEA) to find the minimal characteristic polynomial $\Lambda(z) \in \mathbb{Z}_p[z]$ for the linear recursive sequence $\{f(2^i, 3^i, 5^i, \ldots, \mathfrak{p}_n^i)\}$. We factorize $\Lambda(z)$ over $\mathbb{Z}_p$ to find its roots r_i, where

$$r_i = \mathcal{M}_i(2, 3, 5, \ldots, \mathfrak{p}_n) \tag{1.6}$$

For this to work for each monomial $\mathcal{M}_i(x_1, \ldots, x_n)$ of $f(x_1, \ldots, x_n)$ we need

$$\mathcal{M}_i(2, 3, 5, \ldots, \mathfrak{p}_n) < p.$$

This is a significant restriction in practice. If we determine the total degree $d = \deg(f)$, then $\mathcal{M}_i(2, 3, 5, \ldots, \mathfrak{p}_n) \leq \mathfrak{p}_n^d$. We find the monomials $\mathcal{M}_i(x_1, \ldots, x_n)$ of f by trial division of r_i by the primes $2, 3, \ldots, \mathfrak{p}_n$. For example, suppose that we are interpolating some $f \in \mathbb{Z}[x_1, x_2, x_3]$ and the algorithm produces one of the roots $r_i = 60$, then the trial division with primes $2, 3, 5$ yields the factorization $60 = 2^2 \cdot 3 \cdot 5$ and we obtain the corresponding monomial $\mathcal{M}_i = x_1^2 x_2 x_3$.
Next, we solve for the unknown coefficients a_i Zippel's Vandermonde transpose solver [21]. We have $f(2^i, 3^i, \ldots, \mathfrak{p}^i) = \sum_{j=1}^{t} a_j m_j^i$ for $0 \leq i \leq t$, provided $r_i \neq r_j$ it has a unique solution; the requirement $p > r_i$ ensures this.

1.7 Sparse multivariate rational function interpolation using Kaltofen Yang

Let $h(x_1, \ldots, x_n) = \dfrac{f(x_1, \ldots, x_n)}{g(x_1, \ldots, x_n)} \in \mathbb{Q}(x_1, \ldots, x_n)$, be a multivariate sparse rational function that we want to interpolate, $\mathcal{B}$ be the modular black box for $h = \dfrac{f}{g}$. The Kaltofen Yang algorithm [14] returns $\mu f(x_1, \ldots, x_n)$ and $\mu g(x_1, \ldots, x_n) \in \mathbb{Q}[x_1, \ldots, x_n]$ for some scalar $\mu \in \mathbb{Z}_p$ as output.
We mentioned at the beginning of this chapter that in order to split h, into its numerator f and denominator g Kaltofen and Yang introduced a ring morphism ϕ_1 using the point $\sigma, \beta \in \mathbb{F}^n$. We now state how Kaltofen and Yang defined ϕ_1,

$$\phi_1 \ : \ \mathbb{F}[x_1, x_2, \ldots, x_n, \sigma_1] \longrightarrow \mathbb{F}[x, \sigma_1, \ldots, \sigma_n]$$

$$
\begin{aligned}
x_1 &\longmapsto x, \\
x_j &\longmapsto \beta_j\,(x-\sigma_1)+\sigma_j \quad \text{for all } 2 \le j \le n, \\
\sigma_1 &\longmapsto \sigma_1.
\end{aligned}
$$

$$
\phi_1(h(x_1,\ldots,x_n,\sigma_1)) = h(x,\, \beta_2(x-\sigma_1)+\sigma_2,\, \ldots,\, \beta_n(x-\sigma_1)+\sigma_n,\, \sigma_1)
$$

$$
T(x) = \phi_1(h) = \frac{\phi_1(f)}{\phi_1(g)} = \frac{f(x,\, \beta_2(x-\sigma_1)+\sigma_2,\, \ldots,\, \beta_n(x-\sigma_1)+\sigma_n,\, \sigma_1)}{g(x,\, \beta_2(x-\sigma_1)+\sigma_2,\, \ldots,\, \beta_n(x-\sigma_1)+\sigma_n,\, \sigma_1)} = \frac{\hat{f}(x)}{\hat{g}(x)}
$$

Geometric interpretation of Kaltofen Yang

We can express ϕ_1 as an equation of straight line parametrized by the first coordinate x, passing through σ when $x = \sigma_1$, in the direction of β, in n dimensional affine space $\mathbb{F}^n$ as,

$$
\begin{bmatrix} 0 \\ \sigma_2 - \beta_2\sigma_1 \\ \vdots \\ \sigma_n - \beta_n\sigma_1 \end{bmatrix} + x \begin{bmatrix} 1 \\ \beta_2 \\ \vdots \\ \beta_n \end{bmatrix}. \tag{1.7}
$$

In practice, Kaltofen Yang algorithm uses multiple instances of the map ϕ_1 defined by varying σ, however this fact is hard to capture using their notation stated above. In our implementation, we use $\sigma_i = [\sigma_1^i, \ldots, \sigma_n^i]^T$ which explicitly captures how σ is varied.

To explore the geometric properties of the ring morphism ϕ_1 for the points σ_i, β we define a morphism of affine spaces $\Psi_{\sigma^i,\beta}$ from the corresponding geometric one dimensional affine space $\mathbb{F}^1$ to n dimensional affine space $\mathbb{F}^n$,

$$
\begin{aligned}
\Psi_{\sigma^i,\beta} : \mathbb{F}^1 &\longrightarrow \mathbb{F}^n \\
x &\longmapsto \Big(x,\, \beta_2(x-\sigma_1^i)+\sigma_2^i,\, \ldots,\, \beta_n(x-\sigma_1^i)+\sigma_n^i\Big),
\end{aligned}
$$

comparing it with the equation of a straight line parametrized by the first coordinate x, passing through σ^i when $x = \sigma_1^i$, in the n dimensional affine space $\mathbb{F}^n$ gives us

$$
\mathrm{Im}(\Psi_{\sigma^i,\beta}(x)) = \begin{bmatrix} 0 \\ \sigma_2^i - \beta_2\sigma_1^i \\ \vdots \\ \sigma_n^i - \beta_n\sigma_1^i \end{bmatrix} + x \begin{bmatrix} 1 \\ \beta_2 \\ \vdots \\ \beta_n \end{bmatrix}. \tag{1.8}
$$

Thus, we can see that by varying i, we get a family of parallel affine lines $\mathcal{L}_i$ parametrized by their first coordinate x, passing through σ^i when $x = \sigma_1^i$, in the direction of $\beta \in \mathbb{F}^n$.
In practice, in the Kaltofen Yang algorithm we choose β randomly, once at the beginning of the algorithm, and use the same β throughout. We only vary σ^i by increasing the value

of i. Our geometric notation defined using $\Psi_{\sigma^i,\beta}$ captures this cleanly and we can see that by varying the point σ^i, we get a new line $\mathcal{L}_i$ passing through σ_i, in the direction specified by β.

We will also explore the algebraic properties of the pullback of the geometrical morphism $\Psi_{\sigma^i,\beta}$, the ring morphism $\Psi^*_{\sigma^i,\beta} : \mathbb{F}[x_1, \ldots, x_n] :\longrightarrow \mathbb{F}[x]$.

We then move from $\Psi_{\sigma^i,\beta}$, a map between affine spaces, to $\overline{\Psi}_{\sigma^i,\beta}$, a map between objects in spaces, which in this case is our line $\mathcal{L}_i$, we do so by corestricting $\Psi_{\sigma^i,\beta}(x)$ to its image $\mathrm{Im}(\Psi_{\sigma^i,\beta}) = \mathcal{L}_i \subset \mathbb{F}^n$, as

$$\overline{\Psi}_{\sigma^i,\beta} : \mathbb{F}^1 \longrightarrow \mathcal{L}_i \tag{1.9}$$

We will also look at the algebraic properties of $\overline{\Psi}_{\sigma^i,\beta}$, which is the pullback map of $\overline{\Psi}_{\sigma^i,\beta}$, the ring morphism from the coordinate ring of $\mathcal{L}_i$ to the coordinate ring of $\mathbb{F}$ defined as, $\overline{\Psi}^*_{\sigma^i,\beta} : \mathbb{F}[\mathcal{L}_i] \longrightarrow \mathbb{F}[x]$.

In our implementation of the Kaltofen Yang algorithm, we evaluate the modular black box $\mathcal{B} : (\mathbb{F}^n, p) \longrightarrow \mathbb{F}_p$ of the multivariate rational function $h = f/g \in \mathbb{F}(x_1, \ldots, x_n)$ that we want to interpolate, at points $\overline{\Psi}_{\sigma^i,\beta}(\alpha_j) \in \mathbb{Z}_p^n$ where $\alpha_j \in \mathbb{Z}$ are randomly selected. Geometrically, we are restricting h to $\mathcal{L}_i$, this restriction $h|_{\mathcal{L}_i}$, is expressed in terms of function composition,

$$h|_{\mathcal{L}_i} = h \circ \overline{\Psi}_{\sigma^i,\beta}(x) = \left(\frac{f}{g}\right) \circ \overline{\Psi}_{\sigma^i,\beta}(x) \; = \frac{f\Big(\overline{\Psi}_{\sigma^i,\beta}(x)\Big)}{g\Big(\overline{\Psi}_{\sigma^i,\beta}(x)\Big)} = \frac{\hat{f}_i(x)}{\hat{g}_i(x)} \; \in \; \mathbb{Z}_p(x).$$

This notation is equivalent to the notation used by Kaltofen and Yang, that uses ϕ_1,

$$T(x) = \phi_1(h) = \frac{\phi_1(f)}{\phi_1(g)} = \frac{f(x,\, \beta_2(x-\sigma_1)+\sigma_2,\, \ldots,\, \beta_n(x-\sigma_1)+\sigma_n,\, \sigma_1)}{g(x,\, \beta_2(x-\sigma_1)+\sigma_2,\, \ldots,\, \beta_n(x-\sigma_1)+\sigma_n,\, \sigma_1)} = \frac{\hat{f}(x)}{\hat{g}(x)}$$

However, our notation captures the working of the algorithm in a finer detail.

Combining the two notations

$$T_i(x) = \; h|_{\mathcal{L}_i} = \left(\frac{f}{g}\right) \circ \overline{\Psi}_{\sigma^i,\beta}(x) \; = \frac{f\Big(\overline{\Psi}_{\sigma^i,\beta}(x)\Big)}{g\Big(\overline{\Psi}_{\sigma^i,\beta}(x)\Big)} = \frac{f(x,\, \beta_2(x-\sigma_1^i)+\sigma_2^i,\, \ldots,\, \beta_n(x-\sigma_1^i)+\sigma_n^i)}{g(x,\, \beta_2(x-\sigma_1^i)+\sigma_2^i,\, \ldots,\, \beta_n(x-\sigma_1^i)+\sigma_n^i)} = \frac{\hat{f}_i(x)}{\hat{g}_i(x)}.$$

We can verify that our notation preserves the key property of ϕ_1 that,

$$T_i(\sigma_1^i) = \; h|_{\mathcal{L}_i} = h \circ \overline{\Psi}_{\sigma^i,\beta}(\sigma_1^i) = \left(\frac{f}{g}\right) \circ \overline{\Psi}_{\sigma^i,\beta}(\sigma_1^i) = \frac{f(\sigma_1^i, \ldots, \sigma_n^i)}{g(\sigma_1^i, \ldots, \sigma_n^i)} = \frac{\hat{f}(\sigma_1^i)}{\hat{g}(\sigma_1^i)},$$

By evaluating the modular black box $\mathcal{B}$ of h at a point $\overline{\Psi}_{\sigma^i,\beta}(\alpha_j)$ on the line $\mathcal{L}_i$, we compute

$$T_i(\alpha_j) = h\Big(\overline{\Psi}_{\sigma^i,\beta}(\alpha_j)\Big) = \frac{f\Big(\overline{\Psi}_{\sigma^i,\beta}(\alpha_j)\Big)}{g\Big(\overline{\Psi}_{\sigma^i,\beta}(\alpha_j)\Big)} = \frac{\hat{f}_i(\alpha_j)}{\hat{g}_i(\alpha_j)} \ \in \ \mathbb{Z}_p.$$

Since the restriction of h to the affine line $\mathcal{L}_i$ is a univariate rational function, we can recover $\mu\hat{f}_i(x)$ and $\mu\hat{g}_i(x) \in \mathbb{Z}_p[x]$ from $T_i(x) \in \mathbb{Z}_p(x)$ using any rational function interpolation algorithm. The maximal quotient rational function reconstruction algorithm interpolates a polynomial $u_i(x) \in \mathbb{Z}_p[x]$ from various values of $T_i(\alpha_j)$ we get from evaluating the black box $\mathcal{B}$ at $\overline{\Psi}_{\sigma^i,\beta}(\alpha_j)$, computes $\overline{m}(x) = \prod(x - \alpha_j)$ and uses the extended Euclidean algorithm on $u_i(x), \overline{m}(x)$ to recover $\mu\hat{f}_i(x), \mu\hat{g}_i(x) \mod \overline{m}(x)$ from relation $u_i(x)\hat{g}_i(x) \equiv \hat{f}_i(x) \mod \overline{m}(x)$ for some $\mu \in \mathbb{Z}_p$.
The number of points $\overline{\Psi}_{\sigma^i,\beta}(\alpha_j)$ on the affine line $\mathcal{L}_i$ required to successfully recover $\hat{f}_i(x)$ and $\hat{g}_i(x)$ is $\deg(\hat{f}_i) + \deg(\hat{g}_i) + 2$. Since we do not know $\deg(\hat{f}_i), \deg(\hat{g}_i)$ in advance, at least one extra point is needed to determine the degrees of $\hat{f}_i(x)$ and $\hat{g}_i(x)$ w.h.p.
A critical property of our construction of $\overline{\Psi}_{\sigma^i,\beta}$ from the Kaltofen Yang ring map ϕ_1 is that the following property holds true,

$$\mu\hat{f}_i(\sigma_1^i) = \mu f(\sigma^i) = \mu f(\sigma_1^i, \ldots, \sigma_n^i), \quad \mu\hat{g}_i(\sigma_1^i) = \mu g(\sigma^i) = \mu g(\sigma_1^i, \ldots, \sigma_n^i).$$

This allows us to treat the values $\{\mu\hat{f}_i(\sigma_1^i)\}$ and $\{\mu\hat{g}_i(\sigma_1^i)\}$ as if they came from probing the black boxes of the multivariate numerator μf and denominator μg at the sequence of points $\sigma^1, \sigma^2, \ldots, \sigma^i$.
We then use the Ben-Or Tiwari algorithm to interpolate the numerator $\mu f(x_1, \ldots, x_n)$ and the denominator $\mu g(x_1, \ldots, x_n)$ from $\{\mu\hat{f}_i(\sigma_1^i)\}$ and $\{\mu\hat{g}_i(\sigma_1^i)\}$ respectively.
We use the same setup for the Ben-Or Tiwari algorithm as described in subsection 1.6.2. Our algorithm implementation expects $f(2^i, 3^i, 5^i, \ldots, \mathfrak{p}_n^i) \mod p$, and $g(2^i, 3^i, 5^i, \ldots, \mathfrak{p}_n^i) \mod p$, where $0 \leq i \leq 2\max(t_{num}, t_{den})$ to recover monomials of the numerator and denominator as $\mathcal{M}_{num\ i}(2, 3, 5, \ldots, \mathfrak{p}_n)$ and $\mathcal{M}_{den\ i}(2, 3, 5, \ldots, \mathfrak{p}_n)$ respectively.
We achieve this using $\sigma^i = [2^i, 3^i, 5^i, \ldots, \mathfrak{p}_n^i]$ to define our parametric affine lines $\mathcal{L}_i$ as

$$\overline{\Psi}_{\sigma^i,\beta}(x) = \begin{bmatrix} 0 \\ 3^i - 2^i\beta_2 \\ \vdots \\ \mathfrak{p}_n^i - 2^i\beta_n \end{bmatrix} + x \begin{bmatrix} 1 \\ \beta_2 \\ \vdots \\ \beta_n \end{bmatrix}. \tag{1.10}$$

We recover the coefficients of the numerator and the denominator using Zippel's transposed Vandermonde solver [21].
Thus, in our implementation, we interpolate the numerator $\mu f(x_1, \ldots, x_n)$ and the denominator $\mu g(x_1, \ldots, x_n)$ from $\{\mu\hat{f}_i(2^i)\}$ and $\{\mu\hat{g}_i(2^i)\}$, for $i = 1, 2, 3, \ldots$ respectively.

In order to recover the rational coefficients of h over $\mathbb{Q}$, we use Maple's iratrecon method, which implements Wang's rational reconstruction method starting with 31-bit prime $p = 2^{31} - 1$ and chooses more primes using $prevprime(p)$.

1.8 Application: Solving parametric linear system of equations $Ax = b$.

Consider the parametric linear system $Ax = b$ where $y_1, \ldots, y_m$ constitute the parameter space. Let

$$A \in \mathbb{Z}[y_1, y_2, \ldots, y_m]^{n \times n}$$

be the coefficient matrix and let

$$b \in \mathbb{Z}[y_1, y_2, \ldots, y_m]^n.$$

Assume A has rank n. Then the solution vector $x \in \mathbb{Z}(y_1, \ldots, y_m)^n$. Let

$$x = \begin{bmatrix} x_1 & x_2 & \cdots & x_n \end{bmatrix}^T = \begin{bmatrix} \frac{f_1}{g_1} & \frac{f_2}{g_2} & \cdots & \frac{f_n}{g_n} \end{bmatrix}^T$$

such that for $f_i, g_i \in \mathbb{Z}[y_1, y_2, \ldots, y_m]$, $g_i \neq 0$, and $\gcd(f_i, g_i) = 1$ for $1 \leq i \leq n$. From Cramer's rule, we know that the solutions of $Ax = b$ are given by

$$x_i = \frac{\det(A_i)}{\det(A)} \in \mathbb{Z}(y_1, \ldots, y_m) \implies \det(A_i) = x_i \det(A) \text{ and } g_i \mid \det(A).$$

where A_i is the matrix obtained by replacing the i-th column of A with b, and $\det(A) \in \mathbb{Z}[y_1, y_2, \ldots, y_m]$.

1.8.1 The Bareiss/Edmonds/Lipson algorithm

The Bareiss/Edmonds/Lipson fraction-free Gaussian elimination algorithm [1, 8, 15], triangularizes the augmented matrix $B = [A \mid b]$ to obtain $\det(A) \in \mathbb{Z}[y_1, y_2, \ldots, y_m]$, and then solves the linear system by back substitution.

Expression swell in the Bareiss/Edmonds/Lipson algorithm

The Bareiss/Edmonds/Lipson algorithm computes the leading principal minors $B_{k,k}$(determinant of the principal $k \times k$ sub-matrix of A) . In the last step the determinant of coefficient matrix A is computed using

$$B_{n,n} = \frac{B_{n-1,n-1}B_{n,n} \; - \; B_{n,n-1}B_{n-1,n}}{B_{n-2,n-2}} = \det(A). \tag{1.11}$$

Notice that the numerator in equation 1.11 is the product of the determinants $B_{n,n}$ and $B_{n-2,n-2}$, which can be much larger than $\det(A)$. In $\mathbb{Z}[y_1, \ldots, y_m]$ if $B_{n,n}$ has 1000 terms and $B_{n-2,n-2}$ has 100 terms, the numerators may have 10^5 terms.

1.8.2 Solving system of linear equations using Kaltofen Yang sparse multivariate rational function interpolation.

We will also solve a parametric linear system that arises from the rational B-spline [18] in five parameters $y_1, \ldots, y_5$ using Kaltofen Yang. From Cramer's rule, the solutions to the linear system

$$Ax = b, \qquad A_{ij} \in \mathbb{Z}[y_1, \ldots, y_m], \quad b_i \in \mathbb{Z}[y_1, \ldots, y_m],$$

can be expressed as

$$x_i = \frac{\det(A_i)}{\det(A)}.$$

where both f_i and $\det(A)$ are multivariate polynomials in $\mathbb{Q}[y_1, \ldots, y_m]$, and A_i denotes the matrix obtained from A by replacing its i-th column with the vector b.

The polynomials f_i and $\det(A)$ can be computed using standard polynomial determinant algorithms, such as the Bareiss–Edmonds fraction-free algorithm. The final step is to compute

$$h_i = \gcd(f_i, \det(A))$$

in order to obtain the reduced solution

$$x_i = \frac{f_i/h_i}{\det(A)/h_i}.$$

The Kaltofen–Yang (KY) algorithm avoids the explicit computation of this greatest common divisor, which is often the most expensive step. Moreover, it avoids the intermediate expression swell that occurs in fraction-free Bareiss,Edmonds,Lipton algorithm, and instead interpolates the reduced solution x_i directly.
Let $A\mathbf{x} = b$ be a linear system such that $A_{n\times n}$ and $\forall\ a_{ij} \in A$, $a_{ij} \in \mathbb{F}[y_1, \ldots, y_m]$, then , the solution

$$x_i = \frac{f_i(y_1, \ldots, y_m)}{g_i(y_1, \ldots, y_m)}$$

defines the coordinate functions of a rational map

$$\begin{aligned} \Phi : \mathbb{F}^m &\dashrightarrow \mathbb{F}^n \\ (a_1, \ldots, a_m) &\longmapsto \left(\frac{f_1(a_1, \ldots, a_m)}{g_1(a_1, \ldots, a_m)}, \ldots, \frac{f_n(a_1, \ldots, a_m)}{g_n(a_1, \ldots, a_m)} \right) \end{aligned} \tag{1.12}$$

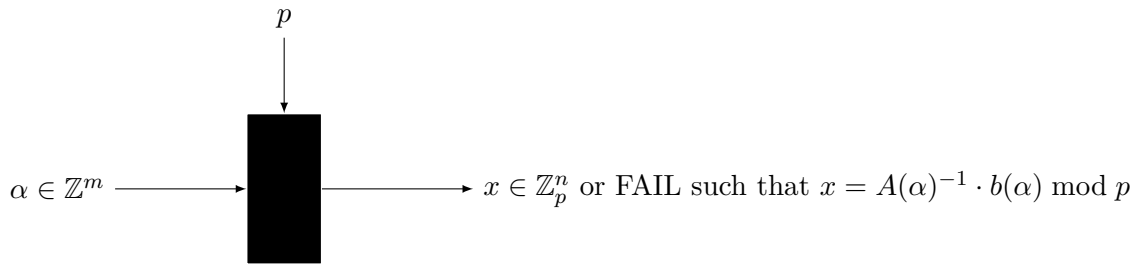
We will see in chapter 5 that applying the Kaltofen Yang algorithm to such a linear system restricts the domain of the rational map to the affine line $\Phi|_{\mathcal{L}_i}$. This implicitly restricts the

coordinate function of the rational map, the solution of the linear system to the affine line $\mathcal{L}_i$ as well.

1.8.3 Modular black box for a parametric system of linear equations

In our setting, the Chinese Remainder Theorem together with rational reconstruction is used to recover the rational coefficients of $x_i \in \mathbb{Q}[y_1, \ldots, y_m]$. Thus, instead of solving a single linear system over $\mathbb{Q}(y_1, \ldots, y_m)$, we solve multiple linear systems over $\mathbb{F}_p$, corresponding to evaluations along distinct affine lines.

Definition 1.8.1. *Let $Ax = b$ be a parametric system of linear equations in m parameters $y_1, \ldots, y_m$ with a non singular coefficient matrix $A \in \mathbb{F}[y_1, \ldots, y_m]^{n \times n}$,such that $\exists$ unique solution $x \in \mathbb{F}(y_1, \ldots, y_m)^n$, that is, $x_k = \dfrac{f_k(y_1, \ldots, y_m)}{g_k(y_1, \ldots, y_m)}$, for $1 \leq k \leq n$. A modular black box $\mathcal{B}$ for $Ax = b$ is a program that on input $\alpha \in \mathbb{F}^m$ and a prime p computes, $x(\alpha) = A^{-1}(\alpha)b(\alpha) \mod p$ (or $\frac{\det(A_i(\alpha))}{\det(A(\alpha))} \mod p$) ie. $\mathcal{B} : (\mathbb{F}^m, p) \longrightarrow \mathbb{F}_p^n$ or outputs FAIL (if $\det(A) \mod p = 0$).*



We give some Maple code for the black box.

```
# Assumptions:
#   A      : n x n Matrix with entries in F[params]
#   b      : length-n Vector with entries in F[params]
#   Vars   : [x1, ..., xn]        (unknowns)
#   params : [y1, ..., ym]        (parameters)

B := proc(alpha::list(integer), p::prime)
    local n, L_alpha, A_alpha, b_alpha, x,m;
    uses LinearAlgebra:-Modular:
    global counter;     counter := counter + 1:
    m:=numelems(params);
    # L_alpha = y_i \gets \alpha_i,  i = 1,...,m
    L_alpha := { seq(params[i] = alpha[i], i = 1 .. m) }:
    # A(alpha) and b(alpha) over F_p
    A_alpha := Eval(A, L_alpha) mod p:
    b_alpha := Eval(b, L_alpha) mod p:

```

```
    # Solve A(alpha) x = b(alpha) in F_p^n
    x := traperror(LinearSolve(p, A_alpha, b_alpha)):
    if x = "matrix is singular" then
        return FAIL
    fi:
    return convert(x, list):
end:
```

Chapter 2

Maximal Quotient Rational Function Reconstruction

Let $T_i(x)$ be a rational function in $\mathbb{F}_p(x)$. As mentioned briefly in chapter 1, the maximal quotient rational function reconstruction algorithm of [17] interpolates a polynomial $u_i(x) \in \mathbb{F}_p[x]$ such that $u_i(\alpha_j) = T_i(\alpha_j)$ for all j obtained by evaluating the modular black box $\mathcal{B}$ of the rational function h at points $\overline{\Psi}_{\sigma^i,\beta}(\alpha_j)$ on the affine line $\mathcal{L}_i$. The algorithm computes $\overline{m}(x) = \prod(x - \alpha_j)$ and then uses the extended Euclidean algorithm with inputs $u_i(x), \overline{m}(x)$ to recover $\mu\hat{f}_i(x), \mu\hat{g}_i(x) \mod \overline{m}(x)$ from relation $u_i(x)\hat{g}_i(x) \equiv \hat{f}_i(x) \mod \overline{m}(x)$ for some $\mu \in \mathbb{F}_p$.
The number of points on the affine line $\mathcal{L}_i$ required to successfully interpolate $\hat{f}_i(x)$ and $\hat{g}_i(x)$ is $\deg(\hat{f}_i) + \deg(\hat{g}_i) + 2$. Since we do not know $deg(\hat{f}_i), deg(\hat{g}_i)$ in advance, at least one extra point is needed to determine the degrees of $\hat{f}_i(x)$ and $\hat{g}_i(x)$ with high probability (w.h.p).

2.1 The extended Euclidean algorithm

The extended Euclidean algorithm is one of the most versatile algorithms in Computer Algebra. The classical Euclidean algorithm is an iterative algorithm defined on Euclidean domains (both integers and univariate polynomials with coefficients in a field), which accepts two non-zero inputs (integers or univariate polynomials over a field) and returns their greatest common divisor (gcd). The extended Euclidean algorithm is a variation of the Euclidean algorithm that computes the coefficients that express the computed gcd as a linear combination of the inputs.
We are interested in applying extended Euclidean algorithm to univariate polynomials. We present the pseudocode of the extended Euclidean algorithm for univariate polynomials over $\mathbb{F}[x]$, where $\mathbb{F}$ is a field, for non-zero inputs $\hat{f}(x), \hat{g}(x) \in \mathbb{F}[x]$ such that $\deg(\hat{f}(x)) = n \geq \deg(\hat{g}(x)) = m$. The extended Euclidean algorithm returns $r_l(x), s_l(x), t_l(x) \in \mathbb{F}[x]$ after performing $O(mn)$ field operations in $\mathbb{F}$.

Extended Euclidean Algorithm [19]

Input: Polynomials $\hat{f}, \hat{g} \in \mathbb{F}[x]$.
Output: $r_l, s_l, t_l \in \mathbb{F}[x]$ such that $r_l = \gcd(\hat{f}, \hat{g}) = s_l \cdot \hat{f} + t_l \cdot \hat{g}$.
$r_0 \leftarrow f, \quad s_0 \leftarrow 1, \quad t_0 \leftarrow 0$
$r_1 \leftarrow g, \quad s_1 \leftarrow 0, \quad t_1 \leftarrow 1$
$i \leftarrow 1$
while $r_i \neq 0$ **do**
    $q_i \leftarrow r_{i-1}/r_i$ ▷ Quotient
    $r_{i+1} \leftarrow r_{i-1} - q_i r_i$ ▷ Remainder
    $s_{i+1} \leftarrow s_{i-1} - q_i s_i$
    $t_{i+1} \leftarrow t_{i-1} - q_i t_i$
    $i \leftarrow i + 1$
end while
$l \leftarrow i - 1$
return $r_l, s_l, t_l \in \mathbb{F}[x]$

Algorithm 1: Extended Euclidean Algorithm on $\mathbb{F}[x]$

We state the following lemma.

Lemma 2.1.1 (Lemma 3.8 in [19])**.**

$$r_i(x) = s_i(x)\hat{f}(x) + t_i(x)\hat{g}(x) \text{ for } 0 \leq i \leq l+1.$$

Substituting $i = l,$ we obtain Bezout's identity, namely, $r_l = s_l(x)\hat{f}_i(x) + t_l(x)\hat{g}_i(x)$.

2.2 Univariate rational function reconstruction

Let $\mathbb{F}$ be a field, p be a prime and $\mathcal{B} : (\mathbb{F}, p) \longrightarrow \mathbb{F}_p$ be a modular black box for a univariate rational function $T(x) = \frac{\hat{f}(x)}{\hat{g}(x)} \in \mathbb{F}(x)$ where $\hat{f}(x), \hat{g}(x) \in \mathbb{F}[x]$ and $\hat{g}(x) \neq 0$ and $\gcd(\hat{f}(x), \hat{g}(x)) = 1$. Then, we can interpolate $T(x)$ by recovering the numerator $\hat{f}(x)$ and the denominator $\hat{g}(x)$ up to a constant factor $c \in \mathbb{F}$ by applying the extended Euclidean algorithm to $u(x), \overline{m}(x)$ where, $u(x) \in \mathbb{F}[x]/\overline{m}(x)$ is computed by polynomial interpolation (Newton or Lagrange) in $O(n^2)$ field operations in $\mathbb{F}$ at randomly chosen points $\alpha_i \in \mathbb{F}$ with black box evaluations y_i at α_i for $1 \leq i \leq n$. I.e. $y_i = u(\alpha_i) = T(\alpha_i) = \dfrac{\hat{f}(\alpha_i)}{\hat{g}(\alpha_i)} \mod p$ for all distinct $\alpha_i \in \mathbb{F}$ such that $\hat{g}(\alpha_i) \mod p \neq 0$ and $\overline{m}(x) = \prod_{i=1}^{n}(x - \alpha_i)$, where $n = \deg(u) = \deg(\hat{f}) + \deg(\hat{g}) + 1$.

By the Chinese Remainder Theorem, we have the ring isomorphism

$$\prod_{i=1}^{n} \mathbb{F}_p[x]/(x-\alpha_i) \;\cong\; \mathbb{F}_p[x]\Big/\prod_{i=1}^{n}(x-\alpha_i) \;\cong\; \mathbb{F}_p[x]/\overline{m}(x),$$

Let $\alpha = (\alpha_1, \ldots, \alpha_n) \in \mathbb{F}^n$ be such that $\alpha_i \neq \alpha_j$ for all $i \neq j$.
By polynomial interpolation [19], for any set of n values (α_i, y_i), there exists a *unique* polynomial $u(x) \in \mathbb{F}[x]$, with $\deg u < n$ such that $u(\alpha_i) = y_i \quad$ for all $1 \leq i \leq n$.
The evaluation of the black box $y_i = \mathcal{B}(\alpha_i, p) = T(\alpha_i) = \frac{\hat{f}(\alpha_i)}{\hat{g}(\alpha_i)}$ for all $1 \leq i \leq n$ where $\hat{g}(\alpha_i) \neq 0$ for all i. Since $\deg(u(x)) < n$ and $\deg(\overline{m}(x)) = n$, and $u(x) \in \mathbb{F}[x]/\overline{m}(x)$,

$$u(\alpha_i) = \frac{\hat{f}(\alpha_i)}{\hat{g}(\alpha_i)} \implies u(\alpha_i)\hat{g}(\alpha_i) = \hat{f}(\alpha_i) \text{ for all } 1 \leq i \leq n.$$

Since this congruence holds for all $i = 1, \ldots, n$, the Chinese Remainder Theorem implies

$$u(x)\hat{g}(x) \equiv \hat{f}(x) \pmod{\overline{m}(x)}. \tag{2.1}$$

By Lemma 2.1.1, we have

$$\overline{m}(x)s_i(x) + u(x)t_i(x) = r_i(x) \text{ for } 1 \leq i \leq l \tag{2.2}$$

where $r_l(x) = gcd(u(x), \overline{m}(x))$. Taking mod $\overline{m}(x)$ in equation 2.2 gives us,

$$u(x)t_i(x) \equiv r_i(x) \bmod \overline{m}(x) \text{ for } 1 \leq i \leq l$$

$$\implies u(x) \equiv \frac{r_i(x)}{t_i(x)} \bmod \overline{m}(x), \text{ for } 1 \leq i \leq l \tag{2.3}$$

provided $\gcd(t_i, \overline{m}) = 1$. Thus, we can use the extended Euclidean algorithm to recover the numerator $\hat{f}(x)$ and the denominator $\hat{g}(x)$ from $u(x)$ and $\overline{m}(x)$ as $r_i(x)$ and $t_i(x)$ respectively for some iteration $1 \leq i \leq l$. This implies that r_i and t_i are not unique, which aligns with the fact that a rational function $\frac{\hat{f}(x)}{\hat{g}(x)}$ is a member of an equivalence class $[\hat{f}(x), \hat{g}(x)]$ with other viable representatives, thus the extended Euclidean algorithm returns multiple candidates from the equivalence class of $[\hat{f}(x), \hat{g}(x)]$. This raises the question of how to distinguish $\hat{f}(x), \hat{g}(x)$ from the other representatives in their equivalence class, which are recovered as r_i and t_i by the algorithm.
If we know the degree of the numerator $\deg(\hat{f}(x))$ and the degree of the denominator $\deg(\hat{g}(x))$, then we need $n > \deg(\hat{f}(x)) + \deg(\hat{g}(x))$ points α_i such that $\hat{g}(\alpha_i) \neq 0$ to compute sufficiently large $\overline{m}(x)$ and interpolate $u(x)$ and run the extended Euclidean algorithm with $\overline{m}(x), u(x)$ as inputs . If, during the execution of the extended Euclidean algorithm there is a unique pair (r_i, t_i) such that $\deg(r_i) = \deg(\hat{f}(x))$ and $\deg(t_i) = \deg(\hat{g}(x))$ then we can identify $\hat{f}(x)$ and $\hat{g}(x)$ by keeping track of $\deg(r_i)$ and $\deg(t_i)$ in every iteration and

terminate the algorithm once we find $\hat{f}(x)$ and $\hat{g}(x)$ of the rational function $T(x)$. But what if we do not know the degrees of the numerator and the denominator? How many points should we use to compute $\overline{m}(x)$ and $u(x)$ then, and how do we distinguish $\hat{f}(x)$ and $\hat{g}(x)$ from the other members of the equivalence class $[\hat{f}(x), \hat{g}(x)]$ recovered by the algorithm as $r_i(x)$ and $t_i(x)$?

2.3 Maximal quotient rational function reconstruction

Monagan [17] observed that provided that the number of points $n > \deg(\hat{f}(x))+\deg(\hat{g}(x))+1$ and the points α_i are randomly chosen from $\mathbb{F}$, then iteration i that produces the numerator $\hat{f}(x)$ as $r_i(x)$ and the denominator $\hat{g}(x)$ as $t_i(x)$ w.h.p is the one for which the degree of the quotient is maximum. Here is the algorithm.

Maximal Quotient Rational Function Reconstruction [17]

Input: $\bar{m}, u \in \mathbb{F}[x]$ where $\mathbb{F}$ is a field and $\deg(m) > \deg(u) \geq 0$ or $u = 0$ and $\deg(\bar{m}) \geq 1$.

Output: Either $f, g \in \mathbb{F}[x]$ satisfying

$$\frac{f}{g} \equiv u \pmod{\bar{m}}, \quad \gcd(u, g) = \gcd(f, g) = 1, \quad \text{and } \deg(f)+\deg(g)+1 < \deg(\bar{m}),$$

or **FAIL** if no such solution exists.

Remark: The degree requirement $\deg(f)+\deg(g)+1 < \deg(m)$ is met by requiring that one of the quotients q_i in the Euclidean algorithm has degree at least 2.

if $u = 0$ **then**
    return $(f, g) \leftarrow (0, 1)$
end if
$(r_0, r_1) \leftarrow (\bar{m}, u), \quad (t_0, t_1) \leftarrow (0, 1)$
$(f, g) \leftarrow (r_1, t_1), \quad q_{\max} \leftarrow 1$
$i \leftarrow 1$
while $r_i \neq 0$ **do**
    $q_i \leftarrow r_{i-1}$ quo r_i
    if $\deg(q_i) > q_{\max}$ **then**
        $q_{\max} \leftarrow \deg(q_i)$
        $(f, g) \leftarrow (r_i, t_i)$
    end if
    $(r_{i+1}, t_{i+1}) \leftarrow (r_{i-1} - q_i r_i,\ t_{i-1} - q_i t_i)$
    $i \leftarrow i + 1$
end while

18: **if** $q_{\max} \leq 1$ **or** $\gcd(f, g) \neq 1$ **then**
19: **return FAIL**
20: **end if**
21: **return** $(f/\operatorname{lc}(g),\ g/\operatorname{lc}(g))$

Algorithm 2: Maximal Quotient Rational Function Reconstruction Algorithm on $\mathbb{F}[x]$

Complexity

If $\deg(\overline{m}(x)) = n > \deg(u(x))$, it takes $O(n^2)$ operations in $\mathbb{F}$ to recover the numerator $\hat{f}(x)$ and the denominator $\hat{g}(x)$ as $r_i(x)$ and $t_i(x)$ respectively. It takes $O(n^2)$ to interpolate $u(x)$ and $O(n^2)$ to compute $\overline{m}(x)$.

Example 1. *Let $\mathbb{F} = \mathbb{Z}_{19}$ and let $\mathcal{B}$ be the blackbox for the rational function*

$$T(x) = \frac{3x^2 + 1}{x + 7} \in \mathbb{Z}_{19}(x)$$

We want to recover the rational function $T(x)$. Let $\alpha_1 = 1, \alpha_2 = 2, \alpha_3 = 3, \alpha_4 = 4, \alpha_5 = 5$. Then

$$\overline{m} = (x-1)(x-2)(x-3)(x-4)(x-5) = x^5 + 4x^4 + 9x^3 + 3x^2 + 8x + 13,$$

$$\mathcal{B}(1) = 10, \mathcal{B}(2) = 12, \mathcal{B}(3) = 18, \mathcal{B}(4) = 1, \mathcal{B}(5) = 0$$

and the interpolating polynomial

$$u = 17x^4 + 6x^3 + 16x^2 + 18x + 10.$$

Table 2.1 shows the values of q_i, r_i, t_i when the MQRFR is called with inputs $\overline{m}$ and u.

Table 2.1: Extended Euclidean algorithm computations for input polynomials $\overline{m}$ and u (over $\mathbb{Z}_{19}$).

i	q_i	r_i	t_i
0	–	$x^5 + 4x^4 + 9x^3 + 3x^2 + 8x + 13$	0
1	$9x + 6$	$17x^4 + 6x^3 + 16x^2 + 18x + 10$	1
2	$5x^2 + 4x + 9$	$11x^2 + 10$	$10x + 13$
3	$9x + 7$	$16x + 15$	$7x^3 + 9x^2 + 10x + 17$
4	–	0	$13x^4 + 3x^3 + 18x^2 + 15x + 8$

In row 2 of the table 2.1, the degree of the quotient q_2 is maximal when compared to the degree of other q_i's. So we get

$$\frac{r_2}{t_2} = \frac{11x^2 + 10}{10x + 13}, \qquad (d = 5 > \deg(f) + \deg(g) = 3).$$

Making the denominator monic and normalizing the numerator with the leading coefficient of the denominator,

$$T(x) = \frac{10^{-1}r_2}{10^{-1}t_2} = \frac{3x^2 + 1}{x + 7}$$

Chapter 3

Ben-Or Tiwari Interpolation

3.1 Introduction

Let $\mathbb{F}$ be a field of characteristic 0 and $f(x_1, \ldots, x_n) = \sum_{i=1}^{\tau} c_i \mathcal{M}_i(x_1, \ldots, x_n) \in \mathbb{F}[x_1, \ldots, x_n]$ be a polynomial with τ terms that we want to interpolate. The Ben-Or Tiwari algorithm is a two-step multivariate interpolation algorithm that interpolates all the variables of the polynomial simultaneously, as opposed to Zippel's multivariate interpolation algorithm, which interpolates the variables one at a time. The central idea in the first step of the Ben-Or Tiwari multivariate interpolation algorithm is that any polynomial f with τ terms evaluated at increasing powers of a point $\alpha \in \mathbb{F}^n$ forms a recursive sequence $\{f(\alpha^i)\}_{i\geq 0}$ of order τ. Each root of the minimal characteristic polynomial $\Lambda(z) \in \mathbb{F}[z]$ of the sequence $\{f(\alpha^i)\}_{i\geq 0}$ corresponds to a monomial of f.
The minimal characteristic polynomial $\Lambda(z) \in \mathbb{F}[z]$ for the sequence $\{f(\alpha^i)\}_{i\geq 0}$ can be constructed by either solving the linear system using a $\tau \times \tau$ Hankel matrix using Gaussian elimination in $O(\tau^3)$ field operations or by using the Berlekamp Massey algorithm [16] in $O(\tau^2)$ field operations.
The second step of the Ben-Or Tiwari algorithm recovers the coefficients of f by solving a transposed Vandermonde linear system using Zippel's transposed Vandermonde solver [21]. If τ is unknown, the input to the Ben-Or Tiwari interpolation algorithm is a black box that uses the early stopping technique mentioned in Chapter 1 to determine τ.

3.2 Linear recurrence and characteristic polynomial

Definition 3.2.1 (Linear recurrence)**.** *Let $\mathbb{F}$ be a field, $\{a_n\}_{n\geq 0} \subset \mathbb{F}$ be a sequence. A homogeneous linear recurrence of order k is a linear combination of k preceding terms that has the form*

$$a_n = c_0 a_{n-k} + c_1 a_{n-k+1} + \cdots + c_{k-1} a_{n-1} \ \forall \ n \geq k$$

with coefficients $c_0, \ldots, c_{k-1} \in \mathbb{F}$. If k is the smallest possible order, then the linear recurrence is called a ***minimal linear recurrence****.*

Definition 3.2.2 (Characteristic polynomial)**.** *Given a linear recurrence of order k, for a sequence $\{a_n\}$ with coefficients $c_0, \ldots, c_{k-1}$ in $\mathbb{F}$, a characteristic polynomial is a monic polynomial*

$$\Lambda(Z) \in \mathbb{F}[Z] = Z^k - c_{k-1}Z^{k-1} - \cdots - c_1 Z - c_0$$

that encodes the recurrence. For a minimal linear recurrence, the corresponding characteristic polynomial is called the ***minimal characteristic polynomial****.*

From the definition of characteristic polynomial, we can see that if the order and coefficients of a linear recurrence are known, it is trivial to construct the characteristic polynomial. The real challenge lies in recovering a linear recurrence from the terms of a given sequence, since a single sequence can satisfy many linear recurrences. However, for every sequence $\{a_n\}$ there exists a unique minimal linear recurrence, thus a unique minimal characteristic polynomial .

3.2.1 Recovering the minimal characteristic polynomial from a given sequence

Definition 3.2.3 (Hankel matrix)**.** *Given a sequence $\{a_n\}$, the matrix constructed from the terms of $\{a_n\}$*

$$H_m = \begin{bmatrix} a_0 & a_1 & a_2 & \ldots & a_{m-1} \\ a_1 & a_2 & a_3 & \ldots & a_m \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{m-1} & a_m & a_{m+1} & \cdots & a_{2m-1} \end{bmatrix}_{m \times m}$$

is called a Hankel matrix.
The Hankel matrix H_m can be an infinite matrix if infinitely many terms of the sequence are used to construct it.

Theorem 3.2.1 (Kronecker-Hankel Theorem [9])**.** *A sequence of numbers satisfies a linear recurrence relation if and only if the associated infinite Hankel matrix has finite rank. The order of the linear recurrence is equal to the rank of the Hankel matrix.*

Gaussian Elimination of a Hankel Matrix

If we know that $\{a_n\}$ is a recurrence sequence of order k, then we can recover the coefficients of the recurrence $c_0, \ldots, c_{k-1}$ by setting up and solving the following linear system defined by the $k \times k$ Hankel matrix H_k using Gaussian elimination in $O(k^3)$ field operations.

$$H_k \overline{c} = \begin{bmatrix} a_0 & a_1 & \cdots & a_{k-1} \\ a_1 & a_2 & \cdots & a_k \\ \vdots & \vdots & \ddots & \vdots \\ a_{k-1} & a_k & \cdots & a_{2k-1} \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_{k-1} \end{bmatrix} = \begin{bmatrix} c_0 a_0 + c_1 a_1 + \cdots + c_{k-1} a_{k-1} \\ c_0 a_1 + c_1 a_2 + \cdots + c_{k-1} a_k \\ \vdots \\ c_0 a_{k-1} + c_1 a_k + \cdots + c_{k-1} a_{2k-1} \end{bmatrix} = \begin{bmatrix} a_k \\ a_{k+1} \\ \vdots \\ a_{2k} \end{bmatrix}$$

This gives us the minimal characteristic polynomial

$$Z^k - c_{k-1}Z^{k-1} - \cdots - c_1 Z - c_0 = 0$$

3.2.2 Polynomials as linear recurrences.

Lemma 3.2.2. *Let $\mathbb{F}[x]$ be a ring of polynomials over field $\mathbb{F}$, let $f(x) \in \mathbb{F}[x]$, and let $\alpha \in \mathbb{F}$ be some arbitrary element of $\mathbb{F}$. Then the sequence*

$$\{f(1), f(\alpha), f(\alpha^2), \dots\}$$

is a linear recursive sequence.

Proof. Let

$$f(x) = b_{k-1}x^{k-1} + b_{k-2}x^{k-2} + \cdots + b_1 x + b_0 \in \mathbb{F}[x],$$

Let

$$a_i = f(\alpha^i) = \sum_{j=0}^{k-1} b_j(\alpha^i)^j = \sum_{j=0}^{k-1} b_j(\alpha^j)^i.$$

Hence, each a_i is a linear combination of the geometric sequences $(\alpha^j)^i$ for $j = 0, 1, \dots, k-1$.

For $m \geq k$, the corresponding $m \times m$ Hankel matrix associated with the sequence $\{a_i\}$ is

$$H_m = \begin{bmatrix} a_0 & a_1 & a_2 & \cdots & a_{m-1} \\ a_1 & a_2 & a_3 & \cdots & a_m \\ a_2 & a_3 & a_4 & \cdots & a_{m+1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{m-1} & a_m & a_{m+1} & \cdots & a_{2m-2} \end{bmatrix}.$$

Using Rank decomposition, we can factorize the Hankel matrix as follows [4],

$$H_m = V_m^T D V_m$$

where,

1. Transposed Vandermonde-type matrix V_m^T:

$$V_m^T = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ \alpha^0 & \alpha^1 & \alpha^2 & \cdots & \alpha^{k-1} \\ (\alpha^0)^2 & (\alpha^1)^2 & (\alpha^2)^2 & \cdots & (\alpha^{k-1})^2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ (\alpha^0)^{m-1} & (\alpha^1)^{m-1} & (\alpha^2)^{m-1} & \cdots & (\alpha^{k-1})^{m-1} \end{bmatrix}_{m \times k}.$$

2. Diagonal matrix D:

$$D = \text{diag}(b_0, b_1, \ldots, b_{k-1}) = \begin{bmatrix} b_0 & 0 & 0 & \cdots & 0 \\ 0 & b_1 & 0 & \cdots & 0 \\ 0 & 0 & b_2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & b_{k-1} \end{bmatrix}_{k \times k}.$$

3. Vandermonde-type matrix V_m:

$$V_m = \begin{bmatrix} 1 & \alpha^0 & (\alpha^0)^2 & \cdots & (\alpha^0)^{m-1} \\ 1 & \alpha^1 & (\alpha^1)^2 & \cdots & (\alpha^1)^{m-1} \\ 1 & \alpha^2 & (\alpha^2)^2 & \cdots & (\alpha^2)^{m-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha^{k-1} & (\alpha^{k-1})^2 & \cdots & (\alpha^{k-1})^{m-1} \end{bmatrix}_{k \times m}.$$

We verify that each entry of H_m satisfies

$$a_{p+q} = \sum_{j=0}^{k-1} b_j (\alpha^j)^{p+q} = \sum_{j=0}^{k-1} b_j (\alpha^j)^p (\alpha^j)^q.$$

The (p, q)-entry of $V_m^T D V_m$ is

$$(V_m^T D V_m)_{p,q} = \sum_{j=0}^{k-1} (V_m^T)_{p,j} D_{j,j} (V_m)_{j,q} = \sum_{j=0}^{k-1} b_j (\alpha^j)^p (\alpha^j)^q = a_{p+q}.$$

Thus, $H_m = V_m^T D V_m$.

By rank factorization theorem, since V_m^T is an $m \times k$ matrix, we have $\text{rank}(V_m^T) \leq k$, D is a $k \times k$ diagonal matrix and V_m is a $k \times m$ matrix, thus $\text{rank}(D) \leq k$, $\text{rank}(V_m) \leq k$,

$$\implies \text{rank}(H_m) = \text{rank}(V_m^T D V_m) \leq \text{rank}(V_m) \leq k.$$

which is finite. Therefore, by theorem 3.2.1 we can say that the entries of the matrix H_k, the terms of the sequence $\{f(\alpha^i)\}$ form a linear recurrence.

□

The above argument can also be extended to multivariate polynomials.
Let $f(x_1, \cdots, x_n) \in \mathbb{F}[x_1, \cdots, x_n]$ with t terms. Let

$$f(x_1, \ldots, x_n) = \sum_{j=1}^{t} c_j x_1^{e_j^1} \cdots x_n^{e_j^n} = \sum_{j=1}^{t} c_j \mathcal{M}_j(x_1, \ldots, x_n), \qquad c_j \neq 0. \tag{3.1}$$

where $\mathcal{M}_j(x_1, \ldots, x_n)$ are the monomials of f. Let $\alpha = \begin{bmatrix} \alpha_1 & \alpha_2 & \cdots & \alpha_n \end{bmatrix}^T \in \mathbb{F}^n$. Then

$$f(\alpha_1, \ldots, \alpha_n) = \sum_{j=1}^{t} c_j \prod_{k=1}^{n} \alpha_k^{e_j^k} = \sum_{j=1}^{t} c_j \mathcal{M}_j(\alpha_1, \ldots, \alpha_n)$$

Fix $\alpha_1, \ldots, \alpha_n \in \mathbb{F}$ and define the evaluation sequence

$$a_i := f(\alpha_1^i, \ldots, \alpha_n^i), \qquad i \geq 0. \tag{3.2}$$

Let $\mathcal{M}_j(\alpha_1, \ldots, \alpha_n) = m_j$. Since each monomial $\mathcal{M}_j$ is multiplicative,

$$\mathcal{M}_j(\alpha_1^i, \ldots, \alpha_n^i) = \prod_{k=1}^{n} (\alpha_k^i)^{e_j^k} = \prod_{k=1}^{n} \left(\alpha_k^{e_j^k}\right)^i = \mathcal{M}_j(\alpha_1, \ldots, \alpha_n)^i = m_j^i.$$

Hence the sequence $\{a_i\}$ admits the exponential-sum representation

$$a_i = \sum_{j=1}^{t} c_j m_j^i. \tag{3.3}$$

For $s \geq t$, define the $s \times s$ Hankel matrix associated with the sequence $\{a_i\}$ by

$$H_s = \begin{bmatrix} a_0 & a_1 & a_2 & \cdots & a_{s-1} \\ a_1 & a_2 & a_3 & \cdots & a_s \\ a_2 & a_3 & a_4 & \cdots & a_{s+1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{s-1} & a_s & a_{s+1} & \cdots & a_{2s-1} \end{bmatrix}.$$

Assuming the values $m_1, \ldots, m_t$ are pairwise distinct, the Hankel matrix admits the rank factorization

$$H_s = V_s^T D V_s, \tag{3.4}$$

where the factors are defined below.

1. Transposed Vandermonde-type matrix V_s^T:

$$V_s^T = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ m_1 & m_2 & \cdots & m_t \\ m_1^2 & m_2^2 & \cdots & m_t^2 \\ \vdots & \vdots & \ddots & \vdots \\ m_1^{s-1} & m_2^{s-1} & \cdots & m_t^{s-1} \end{bmatrix}_{s \times t}.$$

2. Diagonal matrix D:

$$D = \text{diag}(c_1, \ldots, c_t) = \begin{bmatrix} c_1 & 0 & \cdots & 0 \\ 0 & c_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & c_t \end{bmatrix}_{t \times t}.$$

3. Vandermonde-type matrix V_s:

$$V_s = \begin{bmatrix} 1 & m_1 & m_1^2 & \cdots & m_1^{s-1} \\ 1 & m_2 & m_2^2 & \cdots & m_2^{s-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & m_t & m_t^2 & \cdots & m_t^{s-1} \end{bmatrix}_{t \times s}.$$

We can verify that, the (u, v) entry of $V_s^T D V_s$ equals

$$\sum_{j=1}^{t} c_j m_j^u m_j^v = \sum_{j=1}^{t} c_j m_j^{u+v} = a_{u+v}$$

which coincides with the (u, v) entry of the Hankel matrix H_s. Just like in the case of univariate polynomials, by the rank factorization theorem, we can say that the $\text{rank}(H_s)$ is bounded above by t, making it finite, therefore by the Kronecker-Hankel theorem 3.2.1 the sequence

$$\{f(1, 1, \ldots, 1), f(\alpha_1, \ldots, \alpha_n), f(\alpha_1^2, \ldots, \alpha_n^2), \ldots, f(\alpha_1^i, \ldots, \alpha_n^i)\}$$

will be a linear recurrence of order t.

3.3 Berlekamp Massey algorithm

The Berlekamp Massey algorithm (BMA) [16] was created by Elwyn Berlekamp to decode BCH codes and Reed-Solomon codes. James Massey applied it to find the shortest linear feedback shift register(LFSR) for a binary sequence. Given 2τ terms of a linear recurrence sequence over an arbitrary field, the BMA can find minimal characteristic polynomial in $O(\tau^2)$ arithmetic operations in $\mathbb{F}$. This is an improvement of an order of magnitude compared to finding the coefficients of the characteristic polynomial by using Gaussian elimination of the Hankel matrix which takes $O(\tau^3)$ arithmetic operations in $\mathbb{F}$.

3.3.1 Equivalence of Berlekamp Massey algorithm and extended Euclidean algorithm

Let $\{s_i\}_{0\leq i<2\tau}$ be a sequence with $s_0, s_1, \ldots, s_{2\tau-1}$, as the first 2τ terms in $\mathbb{F}$. The syndrome polynomial for the sequence $\{s_i\}_{0\leq i<2\tau}$ is defined as, $S(x) = s_0 + s_1x + \cdots + s_{2\tau-1}x^{2\tau-1} \in \mathbb{F}[x]$. Decoding error-correcting codes, involves solving the **key equation** [3]:

$$S(x)\Sigma(x) \equiv \omega(x) \pmod{x^{2\tau}},$$

where: $\Sigma(x)$ is the error locator polynomial, it is the shortest LFSR (minimal characteristic polynomial) that generates the syndrome sequence, $\omega(x)$ is the error evaluator polynomial and 2τ is the error-correcting capability of the code.
The extended Euclidean algorithm can be applied to $x^{2\tau}, S(x)$ to solve the key equation by computing $s(x), t(x) \in \mathbb{F}[x]$ such that:

$$s_i(x)x^{2\tau} + t_i(x)S(x) = r_i(x),$$

where $r_l(x) = \gcd(x^{2\tau}, S(x))$. The algorithm stops when the degree of the remainder $r_i(x)$ is less than τ. At this iteration, $t_i(x)$ corresponds to the error locator polynomial $\Sigma(x)$, and $r_i(x)$ corresponds to the error evaluator polynomial $\omega(x)$. Here $t_i(x)$ is the minimal characteristic polynomial for sequence $\{s_i\}_{0\leq i<2\tau}$ represented as $\Lambda(z)$. We then compute the roots of $\Lambda(z)$ by factorizing it using Cantor-Zassenhaus algorithm [5].

Berlekamp–Massey Euclidean Algorithm (BMEA)

Input: Sequence $S = [s_0, s_1, \ldots, s_{2\tau-1}] \in \mathbb{F}^{2\tau}$ of length 2τ, prime number p, indeterminate z
Output: minimal characteristic polynomial $t_i(z) \in \mathbb{F}[z]$ with $\deg(t_i) < \tau$
$n \leftarrow \text{length}(S)/2$
$m \leftarrow 2n - 1$
$r_0 \leftarrow Z^{2n}$
$r_1 \leftarrow \sum_{i=0}^{2n-1} s_{2n-1-i} \cdot z^i$ ▷ Constructing Syndrome polynomial
$t_0 \leftarrow 0, \quad t_1 \leftarrow 1$
$i \leftarrow 1$
while $\deg(r_i) \geq n$ **do** ▷ EEA
    $q_i \leftarrow r_{i-1}/r_i$
    $r_{i+1} \leftarrow r_{i-1} - q_i r_i$
    $t_{i+1} \leftarrow t_{i-1} - q_i t_i$
    $i \leftarrow i + 1$
end while
Normalize to make t_i Monic:

16: $\ell \leftarrow$ leading_coeff(t_i)
17: **return** $\ell^{-1} \cdot t_i$

3.4 Ben-Or and Tiwari's multivariate polynomial interpolation

We have covered all the components of the Ben-Or Tiwari interpolation algorithm, here we describe the algorithm. Let $f(x_1, x_2, \cdots, x_n) \in \mathbb{Z}_p[x_1, x_2, \cdots, x_n]$

$$f(x_1, \ldots, x_n) = \sum_{j=1}^{t} c_j x_1^{e_j^1} \cdots x_n^{e_j^n} = \sum_{j=1}^{t} c_j \mathcal{M}_j(x_1, \ldots, x_n), \qquad c_j \neq 0. \tag{3.5}$$

Where $\mathcal{M}_j(x_1, \ldots, x_n)$ are the monomials of f. Let $p_1, \ldots, p_n$ be distinct primes, $m_j = \mathcal{M}_j(p_1, \ldots, p_n) = p_1^{e_j^1} \cdots p_n^{e_j^n}$, and $a_i = f(p_1^i, \ldots, p_n^i) = \sum_{j=1}^{t} c_j m_j^i$. Let $\Lambda(z)$ be defined as follows:

$$\Lambda(z) = z^\tau + c_{\tau-1} z^{\tau-1} + \cdots + c_0 \in \mathbb{F}[z] = \prod_{k=1}^{\tau} (z - m_k).$$

Theorem 3.4.1. *For a polynomial f in (3.1), and $a_i = f(p_1^i, \ldots, p_n^i)$ with distinct primes $p_1, \ldots, p_n$, the sequence $\{a_i\}_{i \geq 0}$ is linear recurrence such that, $\Lambda(z)$ is the minimal characteristic polynomial of $\{a_i\}_{i \geq 0}$ (Ben-Or and Tiwari, 1988).*

3.5 Pseudocode

Ben-Or Tiwari Algorithm

Input: Modular black box $\mathcal{B} : (\mathbb{Z}^n, p) \longrightarrow \mathbb{Z}_p$ for the target polynomial $f(x_1, \ldots, x_n) \in \mathbb{Z}_p[x_1, \ldots, x_n]$, number of variables n, prime p.
Output: Sparse polynomial $f(x_1, \ldots, x_n) \mod p$ or **FAIL**
$Primes \leftarrow [2, 3, \ldots, p_n] \in \mathbb{Z}^n$
$y \leftarrow [\,]$
$T \leftarrow 4$
y.append($\mathcal{B}([1, \ldots, 1], p)$)
while true **do**
    for $j \leftarrow 1$ **to** T **do**
        y.append($\mathcal{B}([2^j, 3^j, \ldots, p_n^j], p)$)
    end for
    Construct minimal characteristic polynomial via Berlekamp Massey algorithm
    $\Lambda(z) \leftarrow$ Berlekamp Massey Euclidean Algorithm(y), where $y \in \mathbb{F}_p^T$

$t \leftarrow \deg(\Lambda(z))$

Factor $\Lambda(z)$ to find roots using Maple's Cantor-Zassenhaus algorithm

$roots \leftarrow$ ROOTS(Λ)

if $t = |roots|$ and $t < T \in \mathbb{F}_p$ **then**

break

end if

$T \leftarrow 2T$

end while

Recover monomials from roots using trial division

$\mathcal{M} \leftarrow$ get_monomial($roots, Primes, n, vars$)

if $\mathcal{M} =$ FAIL **then**

return FAIL

end if

Recover coefficients via Zippel_Vandermonde_solver

$A \leftarrow$ Zippel_Vandermonde_solver($y, terms, roots, \Lambda(Z), p$)

$f \leftarrow \sum_{i=1}^{terms} A_i \mathcal{M}_i$

Algorithm 3: Ben-Or Tiwari multivariate interpolation algorithm.

To get the coefficients of the multivariate polynomial, we solve a transposed Vandermonde linear system

$$V^{\mathsf{T}}\, c = y,$$

where

$$V^{\mathsf{T}} = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ m_1 & m_2 & \cdots & m_t \\ \vdots & \vdots & & \vdots \\ m_1^{t-1} & m_2^{t-1} & \cdots & m_t^{t-1} \end{bmatrix}, \qquad c = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_t \end{bmatrix}, \qquad y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_t \end{bmatrix}.$$

Where $m_i = \mathcal{M}_i(2, 3, \ldots \mathfrak{p}_n)$ The transposed Vandermonde system, can be solved using Zippels Vandermonde solver in $O(t^2)$. fields operations.

3.6 Example of Ben-Or Tiwari multivariate interpolation

Example 2. *Let $\mathcal{B}$ be the black box for $f(x, y, z) = x^2 + 3xy + 5yz \in \mathbb{Z}_{17}[x, y, z]$ in $lex(x > y > z)$ that we wish to recover using the Ben-Or Tiwari algorithm.*
Since we have three variables, we will evaluate the black box $\mathcal{B}$ at $[2^i, 3^i, 5^i]$, where $0 \leq i \leq n$. We can write

$$f(x, y, z) = \begin{bmatrix} x^2 & xy & yz \end{bmatrix}^T \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix}$$

Finding the monomials of $f(x, y, z)$

$T := 4$, *so the input sequence of Berlekamp Massey will have* $2T = 2.4 = 8$ *terms. Sequence* $\{s\} = \mathcal{B}([2^i, 3^i, 5^i], 17) = \{9, 12, 8, 9, 8, 1, 3, 10\}$, *where* $0 \leq i \leq 7$

Syndrome polynomial $S(x) = 9Z^7 + 12Z^6 + 8Z^5 + 9Z^4 + 8Z^3 + Z^2 + 3Z + 10$ *The minimal*

Table 3.1: Berlekamp Massey algorithm (over $\mathbb{Z}_{17}$).

i	r_i	t_i
0	Z^8	0
1	$9Z^7 + 12Z^6 + 8Z^5 + 9Z^4 + 8Z^3 + Z^2 + 3Z + 10$	1
2	$16Z^6 + 9Z^5 + 8Z^4 + 8Z^3 + 8Z^2 + 5Z + 4$	$15Z + 14$
3	$16Z^5 + 9Z^4 + 8Z^3 + 8Z^2 + 11Z + 8$	$16Z^2 + 8Z + 11$
4	$14Z^2 + 14Z + 4$	$Z^3 + 9Z^2 + 4Z + 14$

characteristic polynomial will be,

$$\Lambda(Z) = Z^3 - 3Z^2 - 13Z - 8 \ mod\ 17 = 14 + 4Z + 9Z^2 + Z^3$$

We factorize the minimal characteristic polynomial using Cantor-Zassenhaus algorithm in maple

$$\Lambda(Z) = (Z - 4)(Z - 6)(Z - 15)$$

The roots of the minimal characteristic polynomial encode the structure of monomials, where each prime component of Prime powers correspond to each variable.

$$2 \longleftrightarrow x$$

$$3 \longleftrightarrow y$$

$$5 \longleftrightarrow z$$

Factorizing the roots of characteristic polynomials

$$4 = 2^2 \longleftrightarrow x^2, \quad 6 = 2.3 \longleftrightarrow xy, \quad 15 = 3.5 \longleftrightarrow yz$$

$$f(x, y, z) = \begin{bmatrix} x^2 & xy & yz \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix}$$

We use Zippel's transpose Vandermonde Solver to find the coefficients of $f(x, y, z)$ *The roots of the minimal characteristic polynomial* $\Lambda(Z) = 4, 6, 15$ *are the monomials evaluated*

at $[2^i, 3^i, 5^i] \in \mathbb{Z}_{17}$*, the evaluation sequence satisfies*

$$s_i = f(2^i, 3^i, 5^i) = c_0\, 4^i + c_1\, 6^i + c_2\, 15^i \pmod{17}.$$

Taking $i = 0, 1, 2$ *yields the Vandermonde system* $Vc = s$ *over* $\mathbb{Z}_{17}$*:*

$$\begin{bmatrix} 1 & 1 & 1 \\ 4 & 6 & 15 \\ 16 & 2 & 4 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} s_0 \\ s_1 \\ s_2 \end{bmatrix} = \begin{bmatrix} 9 \\ 12 \\ 8 \end{bmatrix} \pmod{17},$$

where $16 = 4^2$*,* $2 = 6^2$*, and* $4 = 15^2$ *in* $\mathbb{Z}_{17}$*. Solving this system using Zippel's Vandermonde solver gives us the coefficient* $[c_0, c_1, c_2] = [1, 3, 5]$*, hence* $f(x, y, z) = x^2 + 3xy + 5yz$ *in* $\mathbb{Z}_{17}[x, y, z]$*.*

3.7 Failure probability

Let $\mathbb{F}$ be a field and let $f = \sum_{j=1}^{t} c_j \mathcal{M}_j(x_1, \ldots, x_n)$ be a polynomial with t terms where the coefficients c_j are non-zero in $\mathbb{F}$ and the monomials $\mathcal{M}_j$ are distinct. Let $\alpha \in \mathbb{F}^n$ and let $a_i = f(\alpha_1^i, \alpha_2^i, \ldots, \alpha_n^i)$. Define the $s \times s$ Hankel matrix

$$H_s = \begin{bmatrix} a_0 & a_1 & \cdots & a_{s-1} \\ a_1 & a_2 & \cdots & a_s \\ \vdots & \vdots & & \vdots \\ a_{s-1} & a_s & \cdots & a_{2s-2} \end{bmatrix}.$$

If we are given a black box $\mathcal{B} : \mathbb{F}^n \to \mathbb{F}$ for f, the monomials $\mathcal{M}_j$ and the coefficients c_j are unknown. But t is also unknown. If we knew t we could compute $a_i = \mathcal{B}(\alpha)$ for $0 \le i \le 2t-1$, construct the Hankel matrix H_t and the vector $b = [a_t\ a_{t+1}\ \ldots\ a_{2t-1}]^T$ and solve $H_t x = b$ for x to determine $\Lambda(Z)$ the minimal characteristic polynomial. To determine t, Kaltofen and Lee [12] propose to compute

$$\det H_1,\ \det H_2,\ \det H_3,\ \ldots,\ \det H_s,\ \ldots$$

and stop when $\det H_s = 0$ and use $s - 1$ as a guess for t. Since $\det H_{t+1} = 0$ this must stop when $s = t + 1$. However, it is possible that $\det H_s = 0$ for some $1 \le s \le t$. If this happens, we say the algorithm fails to compute t. Kaltofen and Lee [12] point out that this idea will not work for $f = 2x_1 - x_2 - x_3$, or any f such that $\sum_{i=1}^{t} c_i = 0$, since

$$H_1 = [a_0] = [f(1, 1, \ldots, 1)] = [0].$$

Kaltofen and Lee shift the sequence $\{a_i\}$ to start at $i = 1$ so that $H_1 = [a_1] = [f(\alpha)]$. Let $b_i = f(x_1^i, x_2^i, \ldots, x_n^i)$ and let $a_i = b_i(\alpha)$. Define the $s \times s$ Hankel matrices

$$\widehat{H}_s = \begin{bmatrix} b_1 & b_2 & \cdots & b_s \\ b_2 & b_3 & \cdots & b_{s+1} \\ \vdots & \vdots & & \vdots \\ b_s & b_{s+1} & \cdots & b_{2s-1} \end{bmatrix} \quad \text{and} \quad H_s = \begin{bmatrix} a_1 & a_2 & \cdots & a_s \\ a_2 & a_3 & \cdots & a_{s+1} \\ \vdots & \vdots & & \vdots \\ a_s & a_{s+1} & \cdots & a_{2s-1} \end{bmatrix}.$$

Notice that $H_s = \widehat{H}_s(\alpha)$. Let $D_s = \det(\widehat{H}_s)$. To use the Schwartz-Zippel lemma to bound the probability that $D_s(\alpha) = 0$, Kaltofen and Lee first prove that the polynomial $D_s \not\equiv 0$ for $1 \leq s \leq t$ then claim without proof that $\deg(D_s) \leq s^2 \deg(f)$. Thus if S is a finite subset of $\mathbb{F}$ and α is chosen at random from S^n we have

$$\text{Prob}(\det(H_s) = 0) = \text{Prob}(D_s(\alpha) = 0) \leq \frac{\deg(D_s)}{|S|} \leq \frac{s^2 \deg f}{|S|}.$$

Hence the probability that $\det(H_s) = 0$ for any $1 \leq s \leq t$ is

$$\text{Prob}(\prod_{s=1}^{t} D_s(\alpha) = 0) \leq \frac{\sum_{s=1}^{t} s^2 \deg f}{|S|} = \frac{\frac{1}{6}t(2t+1)(t+1)\deg f}{|S|}.$$

There is a problem with this result of Kaltofen and Lee; the bound is cubic in t. In our implementation we have $\mathbb{F} = \mathbb{Z}_p$ for a prime p and $S = \mathbb{Z}_p$ and we start with the prime $p = 2^{31} - 1$. Suppose the unknown polynomial f has $t = 1000$ terms and $n = 5$ variables with $\deg f = 10$. This is not a very large polynomial. The numerator term in the probability $\frac{1}{6}t(2t+1)(t+1)\deg f = 3,338,335,000$ and denominator $|S| = 2^{31} - 1 = 2,147,483,647$ so that the bound on the probability of failure is 1!

In practice, we have observed that this algorithm does not fail even when f has many thousands of terms. The failure bound is a worst-case bound. It assumes that the polynomials $D_1, D_2, \ldots, D_t$ have the maximum possible number of roots in $\mathbb{F}$.

Recall that a non-zero polynomial f of degree d in $\mathbb{F}[x]$ has at most d roots so if pick α at random from S, $\text{Prob}(f(\alpha) = 0) \leq d/|S|$. But if $\mathbb{F}$ is the finite field with p elements, the average number of roots of a polynomial f of degree d is exactly 1. A similar result holds for multivariate polynomials. If f is a non-zero polynomial in $\mathbb{F}[x_1, \ldots, x_n]$ with $d = \deg f$, the Schwartz-Zippel lemma says f can have at most $d|S|^{n-1}$ roots in $\mathbb{F}$. Hence if we pick α from S^n at random,

$$\text{Prob}(f(\alpha) = 0) \leq \frac{d|S|^{n-1}}{|S|^n} = \frac{d}{|S|}.$$

But again, if $\mathbb{F} = \mathbb{Z}_p$ where p is a prime, the average number of roots in $\mathbb{Z}_p$ is p^{n-1}. Hence for a random f of degree d, since there are p^n choices for α, $\text{Prob}(f(\alpha) = 0) = 1/p$.

Thus if the polynomials $D_s = \det(\widehat{H}_s)$ for $1 \le s \le t$ behave randomly, and $\mathbb{F} = \mathbb{Z}_p$ and α is chosen randomly from $\mathbb{Z}_p^n$, $\text{Prob}(D_s(\alpha) = 0) = 1/p$ and $\text{Prob}(\prod_{s=1}^{t} D(\alpha) = 0)$ would be t/p. For $t = 1000$ and $p = 2^{31} - 1$, $t/p < 10^{-6}$ so the algorithm will determine t with good probability.

To reduce the probability of failure Kaltofen and Lee suggest we compute

$$\text{rank}(H_1),\ \text{rank}(H_2),\ \text{rank}(H_3),\ \ldots\ \text{rank}(H_s),\ \ldots$$

and stop when $\text{rank}(H_s) \le s-2$ and use $\text{rank}(H_s)$ to estimate t. This approximately squares the probability of failure since it fails only if two consecutive Hankel matrices H_{s-1} and H_s are singular for some $2 \le s \le t$. Note, computing the rank of the matrices $H_1, H_2, \ldots, H_{t+2}$ using Gaussian elimination costs $\sum_{s=1}^{t+2} O(s^3) = O(t^4)$ arithmetic operations in $\mathbb{F}$, which is expensive.

To reduce the probability of failure and the cost, we compute

$$\text{rank}(H_2),\ \text{rank}(H_4),\ \text{rank}(H_8),\ \text{rank}(H_{16}),\ \ldots, \text{rank}(H_{2^i}),\ \ldots$$

instead and stop when $\det(H_s) = 0$ and use $\text{rank}(H_s)$ to estimate t. This may double the number of probes to the black box but it reduces the probability of failure to

$$\sum_{i=1}^{\lfloor \log_2 t \rfloor} \frac{(2^i)^2 \deg f}{|S|} < \frac{\frac{4}{3}t^2 \deg f}{|S|}$$

which is quadratic in t. It also reduces the cost of the Gaussian elimination from $O(t^4)$ to $O(t^3)$ arithmetic operations in $\mathbb{F}$.

Chapter 4

Sparse Multivariate Rational function interpolation

Let $h(x_1, \ldots, x_n) = \dfrac{f(x_1, \ldots, x_n)}{g(x_1, \ldots, x_n)} \in \mathbb{Q}(x_1, \ldots, x_n)$, be a multivariate sparse rational function that we want to interpolate, $\mathcal{B}$ be the modular black box for $h = \dfrac{f}{g}$. The Kaltofen Yang algorithm [14] returns $\mu f(x_1, \ldots, x_n)$ and $\mu g(x_1, \ldots, x_n) \in \mathbb{Q}[x_1, \ldots, x_n]$ for some scalar $\mu \in \mathbb{Z}_p$ as output.

4.1 Kaltofen Yang algorithm

The Kaltofen-Yang algorithm [13] uses the maximal quotient rational function reconstruction algorithm (MQRFR) and the Ben-Or Tiwari algorithm. Figure 4.1 provides a high-level overview and organization of all the algorithms that constitute the Kaltofen Yang algorithm.

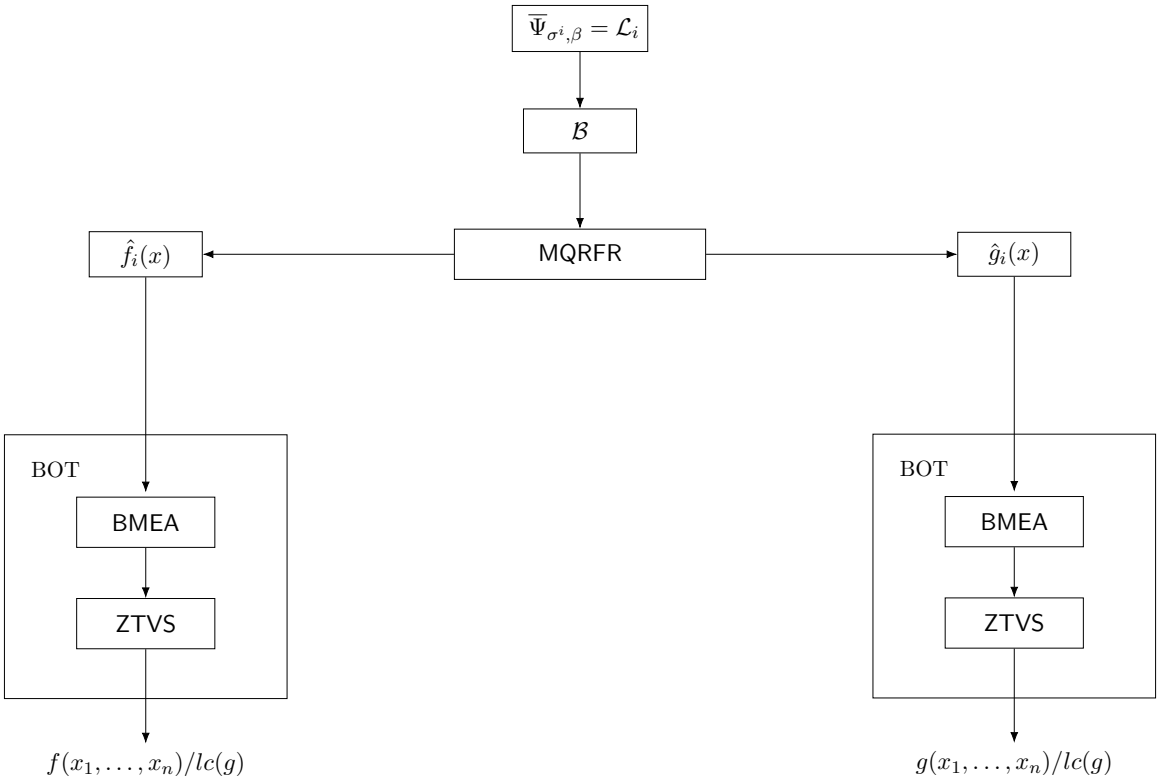


Figure 4.1: High level overview of Kaltofen Yang algorithm

The components of the algorithm are defined as follows:

- $\Psi_{\sigma^i,\beta} : \mathbb{A}^1 \to \mathcal{L}_i \subset \mathbb{A}^n$: The parametric geometric map defining the line $\mathcal{L}_i$ along which we probe the black box.
- $\mathcal{B}$: The black box representing the multivariate rational function h.
- **MQRFR**: The Maximal Quotient Rational Function Reconstruction algorithm, which recovers the univariate rational function restricted to the line.
- $\hat{f}_i(x), \hat{g}_i(x)$: The recovered univariate images of the numerator and denominator, respectively.
- **BOT**: The Ben-Or Tiwari interpolation algorithm.
- **BMEA**: The Berlekamp-Massey algorithm (used within BOT to find the recurrence).
- **ZTVS**: The Zippel Transpose Vandermonde Solver (used within BOT to recover coefficients).

4.1.1 Motivation: The Euclidean Domain Constraint

To understand the architecture presented in Figure 4.1, it is helpful to view the algorithm as bridging the gap between multivariate and univariate algebra. The core reconstruction

tool, MQRFR, relies on the extended Euclidean Algorithm. Consequently, it can only be applied to polynomials over Euclidean domains (such as the univariate polynomial ring $\mathbb{F}[x]$). However, the ring of multivariate polynomials $\mathbb{F}[x_1, \ldots, x_n]$ is *not* a Euclidean domain for $n > 1$. Therefore, we cannot apply MQRFR directly to the multivariate black box.
To overcome this algebraic limitation, Kaltofen and Yang introduce the ring map ϕ_1. We define a morphism $\Psi_{\sigma^i,\beta}$ between affine spaces and show that this map restricts the domain to an affine line $\mathcal{L}_i \cong \mathbb{A}^1$, effectively projecting the non-Euclidean multivariate problem into a Euclidean univariate domain where MQRFR can successfully operate. The subsequent sections of this chapter are dedicated to formalizing this reduction algebraically and explore what it means geometrically.

4.2 Correspondence between Algebra and Geometry

4.2.1 Bridge between computer algebra and algebraic geometry: Computational Reality vs Geometric Intuition

Our implementation of the Kaltofen-Yang algorithm, like many polynomial algorithms in computer algebra, utilizes exact and modular arithmetic over a finite field $\mathbb{F}_p$ for machine-sized primes, with the Chinese Remainder Theorem subsequently employed to reconstruct coefficients in $\mathbb{Q}$ or $\mathbb{Z}$. However, the theoretical framework of algebraic geometry, establishing the correspondence between algebra and geometry, is traditionally formulated over an algebraically closed field.
To reconcile this, we will consider the base field $\mathbb{F}$ to be an algebraic closure while discussing the geometric properties associated with the rational function $h = \dfrac{f}{g} \in \mathbb{F}(x_1, \ldots, x_n)$ and multivariate polynomials $f, g \in \mathbb{F}[x_1, \ldots, x_n]$. Doing so allows us to leverage powerful tools like the Hilbert Nullstellensatz and the theory of birational maps to discuss the mechanics of the Kaltofen Yang algorithm geometrically, even when the actual computations are restricted to $\mathbb{F}_p$.
In the following subsections, we transition from the global study of polynomial coordinate rings $\mathbb{F}[X]$ to the local study of function fields $\mathbb{F}(X)$. This build-up serves as the necessary framework for interpreting the Kaltofen Yang algorithm's mechanics through the lens of birational geometry and dominant rational maps.

4.2.2 Correspondence between polynomials and geometry

Definition 4.2.1 (Affine space)**.** *[6] Given a field $\mathbb{F}$ and a positive integer n, we define the n-dimensional affine space $\mathbb{A}^n$ over $\mathbb{F}$ to be the set*

$$\mathbb{A}^n = \{(a_1, \ldots, a_n) : a_1, \ldots, a_n \in \mathbb{F}\} \tag{4.1}$$

Definition 4.2.2 (Affine variety)**.** *[6] Let $\mathbb{F}$ be an algebraically closed field (an algebraic closure of some base field) and $\mathbb{F}[x_1, \ldots, x_n]$ be the polynomial ring in n variables. Given an ideal $\mathcal{I} \subseteq \mathbb{F}[x_1, \ldots, x_n]$, we define the affine variety $V(\mathcal{I})$ associated with $\mathcal{I}$ as the set of all points in affine space $\mathbb{A}^n$ where every polynomial in the ideal vanishes:*

$$V(\mathcal{I}) = \{a \in \mathbb{A}^n \mid f(a) = 0 \text{ for all } f \in \mathcal{I}\}$$

From a computational perspective, if $\mathcal{I} = \langle f_1, \ldots, f_m \rangle$, then $V(\mathcal{I})$ is simply the solution set of the system of equations $f_1 = \cdots = f_m = 0$.

Example 3. *Let $f = x^2 + y^2 - 1 \in \mathbb{F}[x, y]$, where $\mathbb{F}$ is an algebraically closed field and $\mathcal{I} = \langle f \rangle$ be the principal ideal generated by f. Then*

$V(\mathcal{I}) = \{(x, y) | x, y \text{ are the coordinates of the points on the unit circle with center at } (0, 0)\} \subset \mathbb{A}^2$.

Definition 4.2.3 (The Ideal of a Variety $\mathcal{I}(X)$)**.** *[6] Given any subset $X \subseteq \mathbb{A}^n$, we can define the ideal of all polynomials that vanish on X as:*

$$\mathcal{I}(X) = \{f \in \mathbb{F}[x_1, \ldots, x_n] \mid f(a) = 0 \text{ for all } a \in X\}$$

Theorem 4.2.1 (Hilbert's Nullstellensatz (Strong))**.** *[6] Let $\mathbb{F}$ be an algebraically closed field and $\mathcal{I}$ be an ideal in $\mathbb{F}[x_1, \ldots, x_n]$. Then:*

$$\mathcal{I}(V(\mathcal{I})) = \sqrt{\mathcal{I}}.$$

Definition 4.2.4 (Radical of an ideal)**.** *[6] The radical of an ideal $\mathcal{I} \subset \mathbb{F}[x_1, \ldots, x_n]$ is the set of all polynomials $f \in \mathbb{F}[x_1, \ldots, x_n]$ such that $f^m \in \mathcal{I}$ for some integer $m \geq 1$.*

Hilberts Nullstellensatz establishes a one-to-one correspondence between affine varieties and radical ideals.

Definition 4.2.5 (Prime ideal)**.** *[7] An ideal $\mathcal{I}$ of a commutative ring R is a prime ideal if*

1. *$\mathcal{I}$ is a proper ideal, i.e. $\mathcal{I} \neq R$ (this implies $1 \notin \mathcal{I}$).*
2. *If $ab \in \mathcal{I}$ then $a \in \mathcal{I}$ or $b \in \mathcal{I}$.*

Theorem 4.2.2. *[7] Prime ideals are radical.*

Definition 4.2.6 (Integral domain)**.** *[7] A ring R is an integral domain if it satisfies three conditions:*

1. *Commutativity: Multiplication is commutative ($ab = ba$ for all $a, b \in R$).*

2. *Identity: There exists a multiplicative identity element* $1 \in R$ *such that* $1 \neq 0$.

3. *No Zero Divisors: If the product of two elements is zero, at least one of the elements must be zero, i.e. if* $ab = 0$, *then* $a = 0$ *or* $b = 0$.

Definition 4.2.7 (Field of fraction). *[7] Let* R *be an integral domain, then the field of fractions* $Frac(R)$ *is defined as the set of equivalence classes of ordered pairs* (a, b) *where* $a, b \in R$ *and* $b \neq 0$ *that satisfies the following properties,*

1. *Equivalence Relation: Two pairs* (a, b) *and* (c, d) *are equivalent, denoted* $(a, b) \sim (c, d)$, *if and only if:*$ad = bc$. *The equivalence class of a pair* (a, b) *is typically written as a fraction* $\frac{a}{b}$.

2. *Operations: Addition and multiplication are defined on these equivalence classes to ensure they are well-defined:*
   Addition:
   $$\frac{a}{b} + \frac{c}{d} = \frac{ad + bc}{bd}.$$
   Multiplication:
   $$\frac{a}{b} \cdot \frac{c}{d} = \frac{ac}{bd}.$$

3. *Embedding: There is an injective ring homomorphism* $\iota : R \to Frac(R)$ *defined by* $\iota(a) = \frac{a}{1}$, *which allows* R *to be viewed as a subring of its field of fractions.*

Theorem 4.2.3. *[7] If* $\mathcal{I} \subset \mathbb{F}[x_1, \ldots, x_n]$ *is a prime ideal then* $\mathbb{F}[x_1, \ldots, x_n]/\mathcal{I}$ *is an integral domain.*

Theorem 4.2.4. *[6]* $V(\mathcal{I})$ *is an irreducible variety iff* $\mathcal{I}(V)$ *is a prime ideal.*

Definition 4.2.8 (Regular Functions on an affine variety). *A function* $\rho : V \longrightarrow \mathbb{F}$ *for an affine variety* $V \subset \mathbb{A}^n$ *is regular if there exists a polynomial* $f \in \mathbb{F}[x_1, \ldots, x_n]$ *such that* $\rho(a) = f(a)\ \forall\ a \in V$.

Definition 4.2.9 (Coordinate ring). *[6] Let* $V \subset \mathbb{A}^n$ *be an affine variety, then the coordinate ring* $\mathbb{F}[V]$ *is the ring of all regular functions on* V. *Let* $\mathcal{I}(V)$ *be the ideal of all polynomials vanishing on* V, *then :*

$$\mathbb{F}[V] \cong \mathbb{F}[x_1, \ldots, x_n]/\mathcal{I}(V)$$

Thus, regular functions on an affine variety are polynomials in the coordinate ring.

Definition 4.2.10 (Morphism between affine varieties). *[6] Let* $X \subseteq \mathbb{A}^n$ *and* $Y \subseteq \mathbb{A}^m$ *be affine varieties. A function* $\psi : X \to Y$ *is a morphism if there exist* m *polynomials* $f_1, \ldots, f_m \in \mathbb{F}[x_1, \ldots, x_n]$ *such that for every point* $a = (a_1, \ldots, a_n) \in X$*:*

$$\psi(a) = (f_1(a), f_2(a), \ldots, f_m(a)) \in Y.$$

Definition 4.2.11 (Pullback of a morphism)**.** *[6] Let $X \subseteq \mathbb{A}^n$ and $Y \subseteq \mathbb{A}^m$ be affine varieties, then every morphism $\psi : X \to Y$ induces a pullback map between their coordinate rings*

$$\psi^* : \mathbb{F}[Y] \longrightarrow \mathbb{F}[X].$$

This map is a ring homomorphism. Let $f : Y \longrightarrow \mathbb{F} \in \mathbb{F}[Y]$, be a regular function on Y, then $\psi^*(f) = f \circ \psi$.

Definition 4.2.12 (Isomorphism between affine varieties)**.** *[6] A morphism $\psi : X \to Y$ between affine varieties $X \subseteq \mathbb{A}^n$ and $Y \subseteq \mathbb{A}^m$ is an isomorphism if and only if, there exists an inverse morphism $\phi : Y \to X$ such that $\phi \circ \psi = id_X$ and $\psi \circ \phi = id_Y$.*

Where $\mathrm{id}_X : X \longrightarrow X$ and $\mathrm{id}_Y : Y \longrightarrow Y$ are identity maps on affine varieties X and Y such that $\mathrm{id}_X(a) = a$, $\forall\ a \in X$ and $\mathrm{id}_Y(b) = b$, $\forall\ b \in Y$.

Theorem 4.2.5. *[6] A morphism $\psi : X \to Y$ between affine varieties $X \subseteq \mathbb{A}^n$ and $Y \subseteq \mathbb{A}^m$ is an isomorphism if and only if its pullback $\psi^* : \mathbb{F}[Y] \longrightarrow \mathbb{F}[X]$ is a ring isomorphism.*

4.2.3 Function Field and Localization

Consider the rational function $h = f/g \in \mathbb{F}(x_1, \ldots, x_n)$. The simple act of dividing one polynomial by another removes us from the realm of functions that are well-behaved everywhere. On an affine space $\mathbb{A}^n$, any non-constant denominator g of the rational function h will inevitably vanish somewhere, creating "poles" where the expression becomes undefined. To handle these objects formally, we must first define a mathematical object where these fractions can exist as valid algebraic entities, even if they are not yet defined as functions at every point. This leads us to the construction of the function field.

Definition 4.2.13 (Function field)**.** *Let X be an irreducible variety with $\mathbb{F}[X]$ as its coordinate ring. Then the function field $\mathbb{F}(X)$ of X is the field of fractions of the coordinate ring Frac($\mathbb{F}[X]$). Rational functions are the elements of the function field.*

By moving from the coordinate ring $\mathbb{F}[X]$ to the function field $\mathbb{F}(X)$, we treat f/g as a single algebraic element. However, we now face a discrepancy: we have a vast field of rational functions, yet our previous definition of "regular functions" only covers those that are defined at every point of X.To bridge this gap, we must determine exactly which elements of the function field $\mathbb{F}(X)$ actually "behave" like polynomials across the entire variety. The following lemma uses Hilberts Nullstellensatz to prove that, for affine varieties, the global regular functions are precisely those whose denominators never vanish algebraically, the units of the coordinate ring.

Lemma 4.2.6. *Let X be an affine variety over an algebraically closed field $\mathbb{F}$, and let $\mathbb{F}[X]$ be its coordinate ring. A rational function $h = f/g \in \mathbb{F}[X]$ (where $f, g \in \mathbb{F}[X]$ and $g \neq 0$) is regular on the entire variety X if and only if g is a unit in $\mathbb{F}[X]$.*

Proof. ($\Leftarrow$) Suppose g is a unit in $\mathbb{F}[X]$. By definition, there exists an element $k \in \mathbb{F}[X]$ such that $gk = 1$. In the function field $\mathbb{F}(X)$, we can write:

$$h = \frac{f}{g} = \frac{f \cdot k}{g \cdot k} = \frac{fk}{1} = fk$$

Since f and k are elements of $\mathbb{F}[X]$, their product fk is also in $\mathbb{F}[X]$. Since every element of the coordinate ring defines a regular function on X, h is regular on X.
($\Rightarrow$) Suppose $h = \dfrac{f}{g}$ is regular on the entire variety X. This implies that for every point $x \in X$, $g(x) \neq 0$. Let $\mathcal{J}$ be the ideal in $\mathbb{F}[X]$ generated by all such possible denominators $\{g\}$. Since h is regular at every point, there is no point in X where all these denominators vanish simultaneously. Thus, the zero-set of $\mathcal{J}$ on X is empty:

$$V_X(J) = \emptyset$$

By Hilbert's Weak Nullstellensatz, if $V_X(J) = \emptyset$, then the ideal $\mathcal{J}$ must be the unit ideal:

$$\mathcal{J} = (1) = \mathbb{F}[X]$$

Since $1 \in \mathcal{J}$, the constant function 1 is expressible as a combination of denominators. This implies h itself must coincide with a polynomial in $\mathbb{F}[X]$. □

Lemma 4.2.6 establishes a strict condition: global regularity requires a unit denominator. Consequently, most rational functions f/g are not regular on the entire variety X. However, they are regular on the subset of X where the denominator does not vanish. To study the algebraic properties of functions defined on such subsets, we introduce the concept of localization. Specifically, by formally inverting a single polynomial g, we restrict our focus to the 'principal open set' where $g \neq 0$.

Definition 4.2.14 (Principal Localization)**.** *[7] Let $R = \mathbb{F}[X]$ be the coordinate ring of an affine variety, and let $g \in R$ be a non-zero element. The localization of R at g, denoted by R_g (or $\mathbb{F}[X]_g$), is the ring:*

$$R_g = \left\{ \frac{f}{g^k} \mid f \in R, k \in \mathbb{N} \right\} \subseteq \mathbb{F}(X)$$

Equivalently, R_g can be defined as the quotient ring:

$$R_g \cong R[y]/(gy - 1)$$

where y is a new indeterminate representing the formal inverse g^{-1}.

While the function field $\mathbb{F}(X)$ contains all formal fractions, the localized ring $\mathbb{F}[X]_g$ identifies those functions that are well-behaved specifically where g does not vanish.

Definition 4.2.15 (Principal open set)**.** *For a polynomial $g \in \mathbb{F}[X]$ its principal open set $D(g) = \{x \in X \mid g(x) \neq 0\}$.*

Observe the containment hierarchy: $\mathbb{F}[X] \subseteq \mathbb{F}[X]_g \subseteq \mathbb{F}(X)$. The localization $\mathbb{F}[X]_g$ sits between the global polynomial ring and the full function field, containing precisely those functions whose poles are confined to the zero set $V(g)$.

4.2.4 Correspondence between rational functions and geometry

While regular functions define morphisms that are well-defined everywhere on a variety V, the computational reality of the Kaltofen-Yang algorithm operates within the function field $\mathbb{F}(V)$. To understand the geometry of $\mathbb{F}(V)$, we must look beyond globally defined functions to rational maps.

Definition 4.2.16 (Rational Map)**.** *[6] Let $X \subseteq \mathbb{A}^n$ and $Y \subseteq \mathbb{A}^m$ be affine varieties. A rational map $\phi : X \dashrightarrow Y$ is defined by an m-tuple of rational functions $f_1, \ldots, f_m \in \mathbb{F}(X)$ such that for every point x where all f_i are defined, $\phi(x) = (f_1(x), \ldots, f_m(x)) \in Y$. Unlike a morphism, a rational map is not necessarily defined at every point of X.*

Definition 4.2.17 (Domain of Definition)**.** *[6] The domain of definition of a rational map ϕ, denoted $Dom(\phi)$, is the set of points $x \in X$ where ϕ is regular.If $\phi = (f_1/g_1, \ldots, f_m/g_m)$, the domain of definition contains the intersection of the principal open sets of the denominators:*

$$\bigcap_{i=1}^{m} D(g_i) \subseteq Dom(\phi) \subseteq X$$

Since X is irreducible, this intersection is a non-empty, dense open subset of X.

Definition 4.2.18 (Dominant Rational Map)**.** *[6] A rational map $\phi : X \dashrightarrow Y$ is dominant if the image $\phi(Dom(\phi))$ is dense in Y.*

Definition 4.2.19 (Birational Equivalence)**.** *[6] Two varieties X and Y are birationally equivalent if there exist dominant rational maps $\phi : X \dashrightarrow Y$ and $\psi : Y \dashrightarrow X$ such that $\psi \circ \phi = id_X$ and $\phi \circ \psi = id_Y$ wherever the compositions are defined.*

Theorem 4.2.7 (Category Equivalence)**.** *[10] Two irreducible affine varieties X and Y are birationally equivalent if and only if their function fields are isomorphic:*

$$X \sim_{bir} Y \iff \mathbb{F}(X) \cong \mathbb{F}(Y).$$

Definition 4.2.20 (Graph of a morphism)**.** *For a morphism $\phi : X \longrightarrow Y$, where $X \subseteq \mathbb{A}^n$, and $Y \subseteq \mathbb{A}^m$, the graph of ϕ, denoted as Γ_ϕ is defined as*

$$\Gamma_\phi = \{(x, y) \in X \times Y \mid y = \phi(x)\} \subset \mathbb{A}^{n+m}.$$

If ϕ is a morphism between varieties, then the graph Γ_ϕ is itself a variety (specifically, a closed subvariety of $X \times Y$).

Definition 4.2.21 (Zariski closure)**.** *[6] For any subset $S \subseteq \mathbb{A}^n$, the Zariski closure of S is the vanishing set of the ideal of S. That is:*

$$\overline{S} = V(I(S))$$

4.3 Geometric interpretation of Kaltofen Yang

Now we have everything we need to examine the geometric properties of the Kaltofen-Yang algorithm.

4.3.1 Setup

Kaltofen Yang ring map

Kaltofen and Yang introduce the change in variables through a ring map ϕ_1 defined using points $\beta, \sigma \in \mathbb{A}^n$ as,

$$\begin{aligned}\phi_1 \ : \ \mathbb{F}[x_1, x_2, \ldots, x_n, \sigma_1] &\longrightarrow \mathbb{F}[x, \sigma_1, \ldots, \sigma_n] \\ x_1 &\longmapsto x, \\ x_j &\longmapsto \beta_j\,(x - \sigma_1) + \sigma_j, \qquad 2 \le j \le n. \\ \sigma_1 &\longmapsto \sigma_1\end{aligned}$$

such that

$$\phi_1(h(x_1, \ldots, x_n, \sigma_1)) = h(x,\, \beta_2(x - \sigma_1) + \sigma_2,\, \ldots,\, \beta_n(x - \sigma_1) + \sigma_n, \sigma_1)$$

Kaltofen and Yang also construct the inverse of ϕ_1 as The inverse map

$$\begin{aligned}\phi_1^{-1} \ : \ \mathbb{F}[x, \sigma_1, \ldots, \sigma_n] &\longrightarrow \mathbb{F}[x_1, x_2, \ldots, x_n, \sigma_1] \\ \phi_1^{-1}(x) &\longmapsto x_1, \\ \phi_1^{-1}(\sigma_1) &\longmapsto \sigma_1 \\ \phi_1^{-1}(\sigma_j) &\longmapsto x_j - \beta_j\,(x_1 - \sigma_1) \qquad 2 \le j \le n.\end{aligned}$$

where

$$\phi_1^{-1}(h(x, \sigma_1, \ldots, \sigma_n)) = h(x_1,\, \sigma_1,\, x_2 - \beta_2(x_1 - \sigma_1),\, \ldots,\, x_n - \beta_n(x_1 - \sigma_1))\,.$$

The image explicitly constructs the inverse map ϕ_1^{-1}, which demonstrates that ϕ_1 is bijective. Since ϕ_1 is defined as a ring map (preserving addition and multiplication) and has a valid

two-sided inverse, it is a ring isomorphism. In the subsequent subsections, we will explore this algebraic ring isomorphism geometrically.

Kaltofen Yang ring map ϕ_1 as an affine line

The coordinate transformation $x_j \longmapsto \beta_j(x - \sigma_1) + \sigma_j$ defined by the ring map ϕ_1 admits a natural geometric interpretation. For points $\sigma, \beta \in \mathbb{A}^n$, the coordinate transformation characterized by ϕ_1 can be rewritten as:

$$\beta_j x + (\sigma_j - \beta_j \sigma_1) \text{ where } 2 \leq j \leq n. \tag{4.2}$$

This reformulation reveals that ϕ_1 parameterizes an affine line in $\mathbb{A}^n$. By gathering the coordinates, we obtain the equation of the line in vector form:

$$\mathcal{L}(x) = \begin{bmatrix} 1 \\ \beta_2 \\ \vdots \\ \beta_n \end{bmatrix} x + \begin{bmatrix} 0 \\ \sigma_2 - \beta_2\sigma_1 \\ \vdots \\ \sigma_n - \beta_n\sigma_1 \end{bmatrix}. \tag{4.3}$$

This representation clarifies the geometric roles of the β and σ in ϕ_1: β dictates the direction of the line, while σ determines a specific point on the line such that $\mathcal{L}(\sigma_1) = \sigma$.
In our implementation, $\sigma = [2, 3, \ldots, \mathfrak{p}_n]^T$ and we use positive powers of σ denoted as $\sigma_i = [\sigma_1^i, \ldots, \sigma_n^i]^T$. By substituting σ with σ^i in equation 4.3 we define a line $\mathcal{L}_i$ as,

$$\mathcal{L}_i(x) = \begin{bmatrix} 1 \\ \beta_2 \\ \vdots \\ \beta_n \end{bmatrix} x + \begin{bmatrix} 0 \\ \sigma_2^i - \beta_2\sigma_1^i \\ \vdots \\ \sigma_n^i - \beta_n\sigma_1^i \end{bmatrix}. \tag{4.4}$$

Here, the exponent i acts as a parameter that shifts the base point of the affine line. Keeping $\beta \in \mathbb{A}^n$ fixed, the increasing values of i define a family of parallel lines $\mathcal{L}_i$.
In practice, the Kaltofen-Yang algorithm requires evaluating the black box $\mathcal{B}$ of the rational function h at the image of multiple instances of the map ϕ_1 defined using a fixed point $\beta \in \mathbb{A}^n$ and different points σ, translates geometrically to probing the variety along a sequence of parallel lines $\mathcal{L}_i$ with distinct offsets.

4.3.2 Morphism for the affine line

To rigorously formalize the geometric notion of probing the black box $\mathcal{B}$ of rational function h along a sequence of parallel lines $\mathcal{L}_i$, and to capture the algebraic constraints that arise from these varying base points, we define a corresponding family of morphisms $\{\Psi_{\sigma^i,\beta}\}_i$

where

$$\Psi_{\sigma^i,\beta} \ : \ \mathbb{A}^1 \longrightarrow \mathbb{A}^n$$

$$x \longmapsto (x,\, \beta_2(x-\sigma_1^i)+\sigma_2^i,\, \ldots,\, \beta_n(x-\sigma_1^i)+\sigma_n^i)$$

The image of the map $\Psi_{\sigma^i,\beta}$, denoted $\mathrm{Im}(\Psi_{\sigma^i,\beta}) \subset \mathbb{A}^n$, is the affine line $\mathcal{L}_i$.
The co-restriction of $\Psi_{\sigma^i,\beta}$ to its image is defined as

$$\overline{\Psi}_{\sigma^i,\beta} : \mathbb{A}^1 \twoheadrightarrow \mathcal{L}_i.$$

$\overline{\Psi}_{\sigma^i,\beta}$ is surjective.

The pullback $\Psi^*_{\sigma^i,\beta}(x)$ of the morphism $\Psi_{\sigma^i,\beta}(x)$ is a ring homomorphism given by

$$\Psi^*_{\sigma^i,\beta} \ : \ \mathbb{F}[x_1,\ldots,x_n] \longrightarrow \mathbb{F}[x]$$

$$x_1 \longmapsto x$$
$$x_j \longmapsto \beta_j(x-\sigma_1^i)+\sigma_j^i \qquad 2 \le j \le n.$$

The coordinate ring of the line $\mathcal{L}_i$ is the quotient ring $\mathbb{F}[\mathcal{L}_i] = \mathbb{F}[x_1,\ldots,x_n]/\mathcal{I}(\mathcal{L}_i)$ while the coordinate ring of $\mathbb{F}$ is $\mathbb{F}[x]$. Thus, the pullback of the co-restriction of $\Psi_{\sigma^i,\beta}$ will be

$$\overline{\Psi}^*_{\sigma^i,\beta} : \mathbb{F}[\mathcal{L}_i] \longrightarrow \mathbb{F}[x]$$

Lemma 4.3.1. $\mathcal{L}_i \cong \mathbb{A}^1$.

Proof. $\mathcal{L}_i$ is defined by $\overline{\Psi}_{\sigma^i,\beta}$ such that $\overline{\Psi}_{\sigma^i,\beta}(t) = (t, \beta_2(t-\sigma_1^i)+\sigma_2^i, \ldots, \beta_n(t-\sigma_1^i)+\sigma_n^i)$.
This map sends $t \in \mathbb{A}^1$ to $\mathbb{A}^n$. Let the coordinates of the image in $\mathbb{A}^n$ be $(x_1,\ldots,x_n)$.
Let $\zeta : \mathbb{A}^n \to \mathbb{A}^1$ be defined by:

$$\zeta(x_1,\ldots,x_n) = x_1.$$

This is a polynomial map. We define Φ_i to be the restriction of this projection to the line $\mathcal{L}_i$:

$$\Phi_i = \zeta|_{\mathcal{L}_i}.$$

$\Phi_i \circ \overline{\Psi}_{\sigma^i,\beta} = \mathbf{id}_{\mathbb{A}^1}$

$$(\Phi_i \circ \overline{\Psi}_{\sigma^i,\beta})(t) = \Phi_i(\overline{\Psi}_{\sigma^i,\beta}(t)) = \Phi_i(t,\, \beta_2(t-\sigma_1^i)+\sigma_2^i,\, \ldots,\, \beta_n(t-\sigma_1^i)+\sigma_n^i) = t$$

$$\implies \Phi_i \circ \overline{\Psi}_{\sigma^i,\beta} = \mathrm{id}_{\mathbb{A}^1}.$$

$\overline{\Psi}_{\sigma^i,\beta} \circ \Phi_i = \mathbf{id}_{\mathcal{L}_i}$

Let $(a, \beta_2(a-\sigma_1^i)+\sigma_2^i, \ldots, \beta_n(a-\sigma_1^i)+\sigma_n^i) \in \mathcal{L}_i$, then there exists some $a \in \mathbb{A}^1$ such that $(a, \beta_2(a-\sigma_1^i)+\sigma_2^i, \ldots, \beta_n(a-\sigma_1^i)+\sigma_n^i) = \overline{\Psi}_{\sigma^i,\beta}(a)$.

$$(\overline{\Psi}_{\sigma^i,\beta} \circ \Phi_i)(a, \beta_2(a-\sigma_1^i)+\sigma_2^i, \ldots, \beta_n(a-\sigma_1^i)+\sigma_n^i) = \overline{\Psi}_{\sigma^i,\beta}(\Phi_i(a, \beta_2(a-\sigma_1^i)+\sigma_2^i, \ldots, \beta_n(a-\sigma_1^i)+\sigma_n^i).$$

$$\Phi_i((a, \beta_2(a-\sigma_1^i)+\sigma_2^i, \ldots, \beta_n(a-\sigma_1^i)+\sigma_n^i)) = a.$$

$$\overline{\Psi}_{\sigma^i,\beta}(a) = (a, \beta_2(a-\sigma_1^i)+\sigma_2^i, \ldots, \beta_n(a-\sigma_1^i)+\sigma_n^i).$$

Thus, $\overline{\Psi}_{\sigma^i,\beta} \circ \Phi_i$ is the identity on $\mathcal{L}_i$.
Since $\overline{\Psi}_{\sigma^i,\beta}$ and Φ_i are inverses of each other $\mathcal{L}_i \cong \mathbb{A}^1$. □

Lemma 4.3.2. *The coordinate ring $\mathbb{F}[\mathcal{L}_i] = \mathbb{F}[x_1, \ldots, x_n]/\mathcal{I}(\mathcal{L}_i)$ is an integral domain.*

Proof. For the ring homomorphism

$$\overline{\Psi}^*_{\sigma^i,\beta} \ : \ \mathbb{F}[x_1, \ldots, x_n]/\mathcal{I}(\mathcal{L}_i) \longrightarrow \mathbb{F}[x]$$

$$\mathcal{I}(\mathcal{L}_i) = \ker(\overline{\Psi}^*_{\sigma^i,\beta})$$

From the first theorem of isomorphism for rings,

$$\mathbb{F}[x_1, \ldots, x_n]/\ker(\overline{\Psi}^*_{\sigma^i,\beta}) \cong \mathrm{Im}(\overline{\Psi}^*_{\sigma^i,\beta}).$$

$$\mathrm{Im}(\overline{\Psi}^*_{\sigma^i,\beta}) = \mathbb{F}[x], \text{ therefore } \mathbb{F}[x_1, \ldots, x_n]/\mathcal{I}(\mathcal{L}_i) \cong \mathbb{F}[x]$$

We know that $\mathbb{F}[x]$ is an integral domain; thus, $\mathbb{F}[\mathcal{L}_i] \cong \mathbb{F}[x]$ will also be an integral domain. This aligns with the fact that the line $\mathcal{L}_i$ is an irreducible variety, thus making $\mathcal{I}(\mathcal{L}_i)$ a prime ideal [7], and we know that taking a quotient by a prime ideal results in an integral domain [7]. □

Theorem 4.3.3.

$$\mathrm{Frac}(\mathbb{F}[\mathcal{L}_i]) \cong \mathbb{F}(x).$$

Proof. Since $\mathbb{F}[\mathcal{L}_i]$ and $\mathbb{F}[x]$ are integral domains and $\mathbb{F}[\mathcal{L}_i] \cong \mathbb{F}[x]$ (by the first theorem of isomorphism for rings), by the universal property of localization [7] this isomorphism extends to their fraction fields as well. Thus,

$$\mathrm{Frac}\,\mathbb{F}[\mathcal{L}_i] \cong \mathbb{F}(x).$$

□

Lemma 4.3.4. *Let $g \in \mathbb{F}[x_1, \ldots, x_n]$ be a polynomial of total degree $d \geq 1$. Let $g = g_d + g_{d-1} + \cdots + g_0$ be its decomposition into homogeneous components, where g_k is the*

homogeneous component of degree k. Consider the line $\mathcal{L}_i \subset \mathbb{A}^n$ parameterized by the map $\overline{\Psi}_{\sigma^i,\beta} : \mathbb{A}^1 \to \mathcal{L}_i$ defined as:

$$\overline{\Psi}_{\sigma^i,\beta}(x) = (x,\, \beta_2(x - \sigma_1^i) + \sigma_2^i,\, \ldots,\, \beta_n(x - \sigma_1^i) + \sigma_n^i)$$

where $\beta = [1, \beta_2, \ldots, \beta_n] \in \mathbb{A}^n$ is the direction vector and $\sigma^i = [\sigma_1^i, \ldots, \sigma_n^i] \in \mathbb{A}^n$ is a point such that $\overline{\Psi}_{\sigma^i,\beta}(\sigma_1) = (\sigma_1, \ldots, \sigma_n)$. If $\beta \notin V(g_d)$ (i.e., $g_d(\beta) \neq 0$), then the restricted univariate polynomial $\hat{g}(x) = g(\overline{\Psi}_{\sigma^i,\beta}(x))$ has degree exactly d, and its leading coefficient is $g_d(\beta)$.

Proof.

$$g(\overline{\Psi}_{\sigma^i,\beta}(x)) = g \circ \overline{\Psi}_{\sigma^i,\beta} = \overline{\Psi}^*_{\sigma^i,\beta}(g) = \sum_{k=0}^{d} g_k(\overline{\Psi}_{\sigma^i,\beta}(x)) = \sum_{k=0}^{d} \overline{\Psi}^*_{\sigma^i,\beta}(g_k) = \hat{g}(x).$$

Since $\overline{\Psi}_{\sigma^i,\beta}(x)$ is linear in x, substituting these coordinates into g_k of degree k yields a univariate polynomial in x of degree at most k. Consequently, for all $k < d$:

$$\deg(\Psi^*(g_k)) \leq k < d.$$

Let g_d be written as a sum of monomials:

$$g_d(x_1, \ldots, x_n) = \sum_{w=1}^{\#d} c_w \mathcal{M}_w(x_1, \ldots, x_n)$$

where $c_w \in \mathbb{F}$ are non-zero coefficients, $\#d$ is the number of terms of g with degree d or the number of terms in g_d, and each $\mathcal{M}_w$ is a monomial of degree d, i.e., $\mathcal{M}_w(x_1, \ldots, x_n) = \prod_{j=1}^{n} x_j^{e_{w,j}}$ with $\sum_{j=1}^{n} e_{w,j} = d$.
Applying the pullback $\overline{\Psi}^*_{\sigma^i,\beta}$ to g_d:

$$\overline{\Psi}^*_{\sigma^i,\beta}(g_d) = \sum_{w=1}^{\#d} c_w \overline{\Psi}^*_{\sigma^i,\beta}(\mathcal{M}_w).$$

We know that, $\overline{\Psi}^*_{\sigma^i,\beta}(x_1) = x$ and $\overline{\Psi}^*_{\sigma^i,\beta}(x_j) = \beta_j x - \beta_j \sigma_1^i + \sigma_j^i \quad 2 \leq j \leq n$. We can rewrite $\overline{\Psi}^*_{\sigma^i,\beta}(x_j) = \beta_j x + E_j$ for $1 \leq j \leq n$ where $E_j = \sigma_j^i - \beta_j \sigma_1^i$ From equation 4.4, we can set $\beta_1 = 1, E_1 = 0$. Thus,

$$\overline{\Psi}^*_{\sigma^i,\beta}(\mathcal{M}_w) = \prod_{j=1}^{n} (\beta_j x + E_j)^{e_{w,j}}$$

Expanding each factor $(\beta_j x - \beta_j \sigma_1^i + \sigma_j^i)^{e_{w,j}}$ using the binomial theorem, the highest degree term in x is $(\beta_j x)^{e_{w,j}}$. Therefore, the leading term of the product is:

$$\prod_{j=1}^{n} (\beta_j x)^{e_{w,j}} = x^{(\sum e_{w,j})} \prod_{j=1}^{n} \beta_j^{e_{w,j}} = x^d \mathcal{M}_w(\beta)$$

$$\overline{\Psi}^*_{\sigma^i,\beta}(\mathcal{M}_w) = \mathcal{M}_w(\beta) x^d + (\text{terms of degree } < d)$$

Substituting this back into the summation for $\overline{\Psi}^*_{\sigma^i,\beta}(g_d)$:

$$\overline{\Psi}^*_{\sigma^i,\beta}(g_d) = \sum_{w=1}^{\#d} c_w \left[\mathcal{M}_w(\beta) x^d + O(x^{d-1}) \right]$$

$$\overline{\Psi}^*_{\sigma^i,\beta}(g_d) = \left(\sum_{w=1}^{\#d} c_w \mathcal{M}_w(\beta) \right) x^d + O(x^{d-1})$$

Therefore,

$$\hat{g}(x) = g_d(\beta) x^d + (\text{terms of degree } < d).$$

By the hypothesis since $\beta \notin V(g_d)$, $g_d(\beta) \neq 0$. Therefore, the leading coefficient is non-zero, and the degree of $\hat{g}(x)$ is exactly d. □

Corollary 4.3.4.1. *Let $g(x_1, \ldots, x_n) \in \mathbb{F}[x_1, \ldots, x_n]$ and $\overline{\Psi}_{\sigma^i,\beta} : \mathbb{A}^1 \to \mathcal{L}_i$ defines the line $\mathcal{L}_i$ as:*

$$\overline{\Psi}_{\sigma^i,\beta}(x) = (x,\ \beta_2(x - \sigma_1^i) + \sigma_2^i,\ \ldots,\ \beta_n(x - \sigma_1^i) + \sigma_n^i)$$

then the leading coefficient of $g(\overline{\Psi}_{\sigma^i,\beta}(x))$ depends only on the direction vector β of the affine line $\mathcal{L}_i$.

Proof. From lemma 4.3.4 we have

$$g(\overline{\Psi}_{\sigma^i,\beta}(x) = \hat{g}(x) = g_d(\beta) x^d + (\text{terms of degree } < d).$$

Therefore $\text{lcoeff}(g(\overline{\Psi}_{\sigma^i,\beta}(x))) = \text{lcoeff}(\hat{g}(x))) = g_d(\beta)$. □

This introduces an additional constraint for the Kaltofen Yang algorithm, that the direction vector β that is chosen randomly for the line $\mathcal{L}_i$ should not be in the $V(g_d)$, else we might lose the leading term of our univariate denominators $\hat{g}_i(x)$.

4.4 The working of the Kaltofen Yang Algorithm

We can geometrically reformulate the problem of interpolating a sparse multivariate rational function $h = f/g \in \mathbb{F}(x_1, \ldots, x_n)$ as the reconstruction of the hypersurface $\mathcal{H} \subset \mathbb{A}^{n+1}$, defined as the Zariski closure of the graph of h.

Definition 4.4.1 (Hypersurface)**.** *For a multivariate rational function* $h = \frac{f(x_1,\ldots,x_n)}{g(x_1,\ldots,x_n)} \in \mathbb{F}(x_1,\ldots,x_n)$*, we define the associated hypersurface* $\mathcal{H}$ *as the Zariski closure of the graph:*

$$\mathcal{H} = \Gamma_h = V(x_{n+1} \cdot g(x_1,\ldots,x_n) - f(x_1,\ldots,x_n)) \subset \mathbb{A}^{n+1}.$$

If $\gcd(f,g) = 1$*, then* $x_{n+1}g(x_1,\ldots,x_n) - f(x_1,\ldots,x_n)$ *is irreducible in* $\mathbb{F}[x_1,\ldots,x_n,x_{n+1}]$*. The coordinates of an arbitrary point on* $\mathcal{H}$ *will be of the form*

$$\{(x_1,\ldots,x_n,h(x_1,\ldots,x_n))|(x_1,\ldots,x_n,h(x_1,\ldots,x_n)) \in Dom(h) \times \mathbb{A}^1\} \subset \mathbb{A}^n \times \mathbb{A}^1.$$

Where $Dom(h) = \mathbb{A}^n \setminus V(g)$*.*

Definition 4.4.2 (Sampling Plane)**.** *For the affine line* $\mathcal{L}_i \subset \mathbb{A}^n$ *defined by the morphism* $\overline{\Psi}_{\sigma^i,\beta} : \mathbb{A}^1 \longrightarrow \mathcal{L}_i$ *defined as* $\overline{\Psi}_{\sigma^i,\beta}(x) = (x,\, \beta_2(x-\sigma_1^i)+\sigma_2^i,\, \ldots,\, \beta_n(x-\sigma_1^i)+\sigma_n^i)$*. We define the sampling plane* $\Pi_i \subset \mathbb{A}^{n+1}$ *as the plane over the line* $\mathcal{L}_i$ *extended into the* $(n+1)$*-th dimension:* $\Pi_i = \mathcal{L}_i \times \mathbb{A}^1 \subset \mathbb{A}^{n+1}$*. We parametrize this plane using the map* $\pi_i : \mathbb{A}^2 \to \Pi_i$ *defined by,*

$$\begin{aligned} \pi_i : \mathbb{A}^2 &\longrightarrow \mathcal{L}_i \times \mathbb{A}^1 \subset \mathbb{A}^{n+1} \\ (t,y) &\longmapsto (\overline{\Psi}_{\sigma^i,\beta}(t), y) \end{aligned}$$

Set-theoretically, Π_i *is a 2-dimensional linear subvariety of* $\mathbb{A}^{n+1}$*.*

Finding points on the line $\mathcal{L}_i$

Let $\{\alpha_1,\ldots,\alpha_T\} \subset \mathbb{F}$ be a set of evaluation points. For each α_j, the image $\overline{\Psi}_{\sigma^i,\beta}(\alpha_j) \in \mathbb{A}^n$ represents a specific point on the line $\mathcal{L}_i$.

Evaluating the black box of the rational function h

Probing the black box for the rational function h at $\overline{\Psi}_{\sigma^i,\beta}(\alpha_j)$ is equivalent to computing the restriction of the rational function h at the line $\mathcal{L}_i$ denoted as $h|_{\mathcal{L}_i}$ and evaluating this restriction $h|_{\mathcal{L}_i}$ at α_j, denoted as $h|_{\mathcal{L}_i}(\alpha_j)$.
Algebraically, from theorem 4.3.3, we know that $\mathbb{F}[\mathcal{L}_i] \cong \mathbb{F}(x)$ thus,

$$\overline{\Psi}^*_{\sigma^i,\beta}(h(x_1,\ldots,x_n)) = h|_{\mathcal{L}_i} = h \circ \overline{\Psi}_{\sigma^i,\beta} = \overline{\Psi}^*_{\sigma^i,\beta}\left(\frac{f}{g}\right) = \frac{\overline{\Psi}^*_{\sigma^i,\beta}(f)}{\overline{\Psi}^*_{\sigma^i,\beta}(g)} = \frac{\hat{f}_i(x)}{\hat{g}_i(x)} = T_i(x).$$

Computationally,

$$h|_{\mathcal{L}_i}(\alpha_j) = \left(\frac{f}{g}\right) \circ \overline{\Psi}_{\sigma^i,\beta}(\alpha_j) = \frac{f\left(\overline{\Psi}_{\sigma^i,\beta}(\alpha_j)\right)}{g\left(\overline{\Psi}_{\sigma^i,\beta}(\alpha_j)\right)} = \frac{f(\alpha_j,\, \beta_2(\alpha_j-\sigma_1^i)+\sigma_2^i,\, \ldots,\, \beta_n(\alpha_j-\sigma_1^i)+\sigma_n^i)}{g(\alpha_j,\, \beta_2(\alpha_j-\sigma_1^i)+\sigma_2^i,\, \ldots,\, \beta_n(\alpha_j-\sigma_1^i)+\sigma_n^i)} = \frac{\hat{f}_i(\alpha_j)}{\hat{g}_i(\alpha_j)}.$$

Geometrically, this operation is equivalent to identifying a point on the hypersurface $\mathcal{H}$. In particular, the input-output pair corresponds to the intersection point P_j on the sampling plane Π_i given by:

$$P_j = \pi_i(\alpha_j, h(\overline{\Psi}_{\sigma^i,\beta}(\alpha_j))) \in \Pi_i \cap \mathcal{H}.$$

Recovering $\hat{f}_i(x), \hat{g}_i(x)$

From the set of evaluation points $\{(\alpha_j, h(\overline{\Psi}_{\sigma^i,\beta}(\alpha_j)))\}$ for $1 \leq j \leq T$, we compute the modulus polynomial $\overline{m}(x)$ and the interpolating polynomial $u_i(x)$ such that:

$$\overline{m}(x) = \prod_{j=1}^{T}(x - \alpha_j) \in \mathbb{F}[x] \quad \text{and} \quad u_i(x) = \text{Interpolate}(\alpha_j, h(\overline{\Psi}_{\sigma^i,\beta}(\alpha_j))) \in \mathbb{F}[x].$$

By construction, $u_i(x)$ satisfies the congruence:

$$u_i(x) \equiv \frac{\hat{f}_i(x)}{\hat{g}_i(x)} \pmod{\overline{m}(x)}.$$

To recover the univariate polynomials $\hat{f}_i(x)$ and $\hat{g}_i(x)$, we apply Monagan's Maximal Quotient Rational Function Reconstruction (MQRFR) algorithm. This algorithm uses the Extended Euclidean Algorithm (EEA) with inputs $\overline{m}(x)$ and $u_i(x)$. Recall from lemma 2.1.1 that $s_k\overline{m} + t_k u_i = r_k$ holds true for all the inputs $\overline{m}, u_i$ and remainders r_k. The MQRFR algorithm identifies a specific iteration k (where the degree of the quotient is maximal) such that:

$$t_k(x) \cdot u_i(x) \equiv r_k(x) \pmod{\overline{m}(x)}.$$

The algorithm returns $\mu\hat{f}_i(x) = r_k(x)$ and $\mu\hat{g}_i(x) = t_k(x)$ for some scalar $\mu \in \mathbb{F}$.

Definition 4.4.3 (Intersection Curve $\mathcal{C}_i$)**.** *Let $T_i(x) = \hat{f}_i(x)/\hat{g}_i(x) \in \mathbb{F}(x)$ be the univariate rational function obtained by restricting h to the line $\mathcal{L}_i$. Let $U_i = \mathbb{A}^1 \setminus V(\hat{g}_i)$ be the dense open domain where T_i is regular.*
We define the parametrization map $\gamma_i : U_i \longrightarrow \mathbb{A}^{n+1}$ on the domain U_i as:

$$\gamma_i(t) = \left(\overline{\Psi}_{\sigma^i,\beta}(t),\ T_i(t)\right) = \left(\overline{\Psi}_{\sigma^i,\beta}(t),\ \frac{\hat{f}_i(t)}{\hat{g}_i(t)}\right), \tag{4.5}$$

where $\overline{\Psi}_{\sigma^i,\beta} : \mathbb{A}^1 \longrightarrow \mathcal{L}_i$ is the affine parameterization of the line $\mathcal{L}_i$.
The intersection curve $\mathcal{C}_i \subset \mathbb{A}^{n+1}$ is defined as the Zariski closure of the image of this map:

$$\mathcal{C}_i = \overline{\text{Im}(\gamma_i)} = \overline{\{\gamma_i(t) \mid t \in U_i\}}. \tag{4.6}$$

Geometrically, $\mathcal{C}_i$ corresponds to the graph of the univariate function T_i lifted into the ambient space $\mathbb{A}^{n+1}$, strictly contained within the sampling plane Π_i.

Lemma 4.4.1. *The curve $\mathcal{C}_i \subset \Pi_i$.*

Proof. We demonstrate that the graph morphism γ_i factors through π_i the map of the plane Π_i.
Let $\rho : U_i \subset \mathbb{A}^1 \longrightarrow \mathbb{A}^2$ be the graph morphism defined by:

$$\rho(t) = (t,\ T_i(t)) \in \mathbb{A}^2$$

Recall that the plane Π_i is the image of the morphism $\pi_i : \mathbb{A}^2 \to \mathbb{A}^{n+1}$ defined by $\pi_i(t, u) = (\overline{\Psi}_{\sigma^i,\beta}(t), u)$. We compose π_i with the section map ρ:

$$(\pi_i \circ \rho)(t) = \pi_i(t,\ T_i(t)) = (\overline{\Psi}_{\sigma^i,\beta}(t),\ T_i(t))$$

Observing the right-hand side, we see this is exactly the definition of the curve morphism $\gamma_i(t)$. Thus, we have the factorization:

$$\gamma_i = \pi_i \circ \rho$$

Since γ_i factors through π_i, the image of γ_i must be contained within the image of π_i. Therefore, $\mathcal{C}_i \subset \Pi_i$. □

Theorem 4.4.2. *(Intersection curve) The intersection curve C_i lies on the hypersurface $\mathcal{H}$. Consequently, the curve C_i is contained in the intersection of the hypersurface and the sampling plane:*

$$\mathcal{C}_i \subseteq \mathcal{H} \cap \Pi_i.$$

Proof.

$$\text{Let } K = x_{n+1} \cdot g(x_1, \ldots, x_n) - f(x_1, \ldots, x_n)) \in \mathbb{F}[x_1, \ldots, x_n, x_{n+1}].$$

Recall that the hypersurface $\mathcal{H}$ is defined as the vanishing set of the polynomial K.

$$\mathcal{H} = V(K).$$

To prove that $\mathcal{C}_i \subseteq \mathcal{H}$, it suffices to show that the generic point of the curve satisfies the equation of the hypersurface. Consider an arbitrary point in the image of the parametrization map $\gamma_i(t)$ for $t \in U_i$:

$$P_t = \gamma_i(t) = \left(\overline{\Psi}_{\sigma^i,\beta}(t), T_i(t)\right) \in \mathbb{A}^{n+1}.$$

We evaluate the defining polynomial K at the point P_t. Substituting the coordinates $(x_1, \ldots, x_n) = \overline{\Psi}_{\sigma^i,\beta}(t)$ and $x_{n+1} = T_i(t)$:

$$K(P_t) = T_i(t) \cdot g(\overline{\Psi}_{\sigma^i,\beta}(t)) - f(\overline{\Psi}_{\sigma^i,\beta}(t)).$$

By the definition of the algebraic pullback (Section 4.3), the evaluation of the multivariate polynomials f and g along the line $\mathcal{L}_i$ yields the univariate restricted polynomials $\hat{f}_i(t)$ and $\hat{g}_i(t)$:

$$g(\overline{\Psi}_{\sigma^i,\beta}(t)) = \Psi^*_{\sigma^i,\beta}(g) = \hat{g}_i(t),$$

$$f(\overline{\Psi}_{\sigma^i,\beta}(t)) = \Psi^*_{\sigma^i,\beta}(f) = \hat{f}_i(t).$$

Furthermore, the univariate rational function $T_i(t)$ is defined as the ratio of these pullbacks:

$$T_i(t) = \frac{\hat{f}_i(t)}{\hat{g}_i(t)}.$$

Substituting these back into the evaluation of K:

$$K(P_t) = \left(\frac{\hat{f}_i(t)}{\hat{g}_i(t)}\right) \cdot \hat{g}_i(t) - \hat{f}_i(t).$$

For all $t \in U_i$ (where $\hat{g}_i(t) \neq 0$), the terms cancel exactly:

$$K(P_t) = \hat{f}_i(t) - \hat{f}_i(t) = 0.$$

Thus, every point in the image of γ_i lies in the variety $\mathcal{H}$. Since $\mathcal{H}$ is a Zariski-closed set, the closure of the image, $\mathcal{C}_i = \overline{\mathrm{Im}(\gamma_i)}$, must also be contained in $\mathcal{H}$. Combining this with Lemma 4.4.1 (which states $\mathcal{C}_i \subseteq \Pi_i$), we conclude:

$$\mathcal{C}_i \subseteq \mathcal{H} \cap \Pi_i.$$

□

Figure 4.2 shows the affine line $\mathcal{L}_i$, the sampling plane Π_i, points on the affine line $\overline{\Psi}_{\sigma^i,\beta}(\alpha_1), \overline{\Psi}_{\sigma^i,\beta}(\alpha_2)$, points $P_1 = (\overline{\Psi}_{\sigma^i,\beta}(\alpha_1), h(\overline{\Psi}_{\sigma^i,\beta}(\alpha_1)), P_2 = (\overline{\Psi}_{\sigma^i,\beta}(\alpha_2), h(\overline{\Psi}_{\sigma^i,\beta}(\alpha_2))$ and intersection curve $\mathcal{C}_i = \mathcal{H} \cap \Pi_i$.

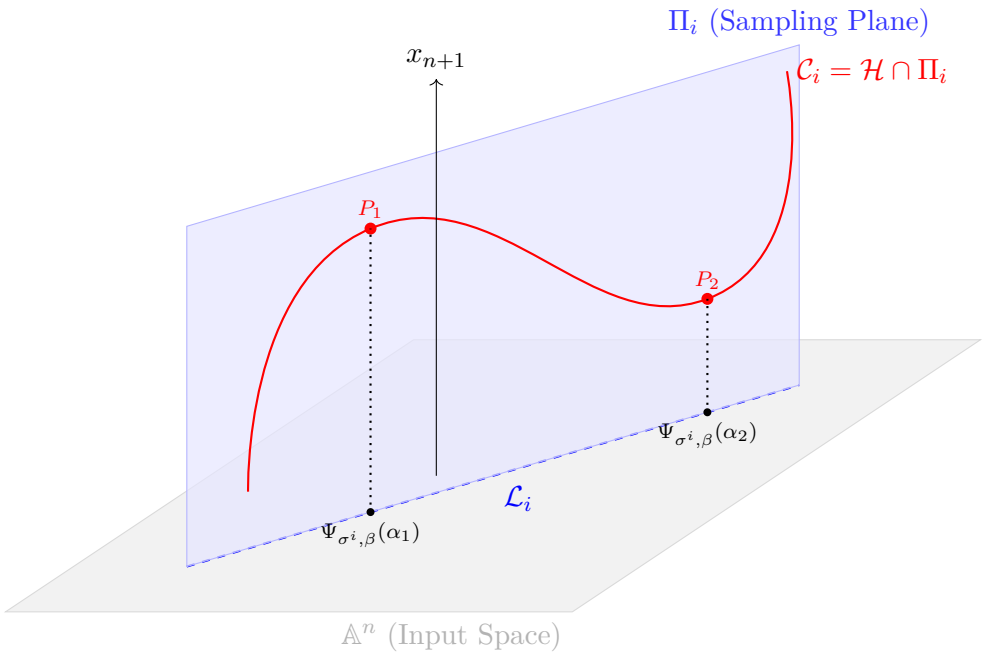


Figure 4.2: Geometric visualization of the restriction step. The black box evaluation at a point $\Psi_{\sigma^i,\beta}(\alpha_j)$ on the line $\mathcal{L}_i$ is equivalent to identifying a point P_j on the intersection curve $\mathcal{C}_i$. This curve is formed by the intersection of the hypersurface $\mathcal{H}$ with the sampling plane Π_i .

Ben-Or Tiwari Interpolation

Because of our construction of $\overline{\Psi}_{\sigma^i,\beta}$ from the Kaltofen Yang ring isomorphism ϕ_1, the following property holds true,

$$\mu\hat{f}_i(\sigma_1^i) = \mu f(\sigma^i) = \mu f(\sigma_1^i, \ldots, \sigma_n^i), \quad \mu\hat{g}_i(\sigma_1^i) = \mu g(\sigma^i) = \mu g(\sigma_1^i, \ldots, \sigma_n^i).$$

This allows us to treat the values $\{\mu\hat{f}_i(\sigma_1^i)\}$ and $\{\mu\hat{g}_i(\sigma_1^i)\}$ as if they came from probing the black boxes of the multivariate numerator μf and denominator μg at the sequence of points $\sigma^1, \sigma^2, \ldots, \sigma^i$. We construct the syndrome sequences:

$$S_f = \{\mu\hat{f}_i(\sigma_1^i)\}_{1\leq i\leq 2\tau} \quad \text{and} \quad S_g = \{\mu\hat{g}_i(\sigma_1^i)\}_{1\leq i\leq 2\tau}.$$

We then apply the Berlekamp-Massey algorithm to these sequences to compute the minimal characteristic polynomials $\Lambda_f(Z), \Lambda_g(Z) \in \mathbb{F}[Z]$ for the numerator f and denominator g, respectively. Finally, we determine the roots of $\{r_{\Lambda_f}\}, \{r_{\Lambda_g}\}$. These roots correspond to the values of the monomials evaluated at the point $\sigma = [\sigma_1, \ldots, \sigma_n]$. We recover the multivariate monomials of f and g by performing trial division of the roots $\{r_{\Lambda_f}\}, \{r_{\Lambda_g}\}$ by $\sigma = \{\sigma_1, \ldots, \sigma_n\}$.
We finally recover the coefficients of f and g by solving the transposed Vandermonde system using Zippel's Vandermonde solver.

Table 4.1: Three perspectives on restricting the rational function h to the line $\mathcal{L}_i$.

Perspective	**Formal Operation**	**Result / Interpretation**
Algebraic (Function Space)	$h\|_{\mathcal{L}_i} = h \circ \overline{\Psi}_{\sigma^i,\beta} = \dfrac{f \circ \overline{\Psi}_{\sigma^i,\beta}}{g \circ \overline{\Psi}_{\sigma^i,\beta}} = \dfrac{\hat{f}_i(x)}{\hat{g}_i(x)}$	A univariate rational function $T_i(x) \in \mathbb{F}(x)$ defined over the function field of the line that is constructed implicitly by the MQRFR algorithm by computing $\hat{f}_i(x), \hat{g}_i(x) \in \mathbb{F}[x]$.
Computational (Pointwise)	$h(\overline{\Psi}_{\sigma^i,\beta}(\alpha_j)) = \dfrac{\hat{f}_i(\alpha_j)}{\hat{g}_i(\alpha_j)}$	A concrete scalar value $h(\overline{\Psi}_{\sigma^i,\beta}(\alpha_j)) \in \mathbb{F}$ obtained by black box evaluation at the point $h(\overline{\Psi}_{\sigma^i,\beta}(\alpha_j))$ on the line $\mathcal{L}_i$.
Geometric (Hypersurface)	$P_j = \pi_i(\alpha_j, h(\overline{\Psi}_{\sigma^i,\beta}(\alpha_j))) \in \Pi_i \cap \mathcal{H}$	A specific point P_j lying on the planar intersection curve $\mathcal{C}_i$ (the slice of the hypersurface $\mathcal{H}$).

4.5 Pseudocode

The Multivariate rational function interpolation algorithm that we present here uses all the algorithms described in the previous chapter.

Multivariate Rational Function Interpolation

1: **Input:** Modular black box $\mathcal{B} : (\mathbb{Z}^n, p) \longrightarrow \mathbb{Z}_p$ for a rational function $\frac{f(x_1,\ldots,x_n)}{g(x_1,\ldots,x_n)}$ over $\mathbb{Z}_p$, where $f, g \in \mathbb{Z}[x_1, \ldots, x_n]$, $\gcd(f, g) = 1$, g is monic, prime p, number of variables n and list of variables $vars$.

2: **Output:** $f(x_1, \ldots, x_n), g(x_1, \ldots, x_n)$ or **FAIL**.

3: $Primes \leftarrow [2, 3, \ldots, p_n] \in \mathbb{Z}^n$

4: Pick a non zero random vector $\beta = [\beta_2, \ldots, \beta_n] \in \mathbb{Z}_p^{n-1}$

5: $num \leftarrow [\,]$, $den \leftarrow [\,]$

6: $num\_points\_mqrfr \leftarrow 0$

7: $numerator\_done \leftarrow$ false, $denominator\_done \leftarrow$ false

8: $T\_old \leftarrow 1, \qquad T \leftarrow 4$

9: $\sigma^0 \leftarrow [1, \ldots, 1] \in \mathbb{Z}_p^n$

10: $(\hat{f}_0, \hat{g}_0) \leftarrow \text{NDSA}(\mathcal{B}, \sigma^0, \beta, p, T)$ where, $\hat{f}_0, \hat{g}_0 \in Z_p[x]$

11: $num.\text{append}(\hat{f}_0(\sigma_1^0) \bmod p), \qquad den.\text{append}(\hat{g}_0(\sigma_1^0) \bmod p)$

12: $num\_points\_mqrfr \leftarrow \deg(\hat{f}_0) + \deg(\hat{g}_0) + 2$

13: **while** true **do**

14: **for** $j \leftarrow T\_old$ to $2T - 1$ **do**

15: $\sigma^j \leftarrow [2^j, 3^j, \ldots, p_n^j] \mod p$

16: $(\hat{f}_j, \hat{g}_j) \leftarrow \text{NDSA}(\mathcal{B}, \sigma^j, \beta, p, num\_points\_mqrfr)$

17: **if** *not numerator_done* **then** $num.\text{append}(\hat{f}_j(\sigma_1^j) \bmod p)$ **end if**

18: **if** *not denominator_done* **then** $den.\text{append}(\hat{g}_j(\sigma_1^j) \bmod p)$ **end if**

19: **end for**

▷ Construct minimal characteristic polynomials using Berlekamp-Massey algorithm

20: **if** *not numerator_done* **then**

21: $\Lambda_{num}(Z) \leftarrow \text{Berlekamp\_Massey}(num, p, z) \in \mathbb{Z}_p[Z]$

22: $s \leftarrow \deg(\Lambda_{num}(Z))$

23: Let $roots_{num}$ be the distinct roots of $\Lambda_{num}(Z) \in \mathbb{Z}_p[Z]$.

24: **end if**

25: **if** *not denominator_done* **then**

26: $\Lambda_{den}(Z) \leftarrow \text{Berlekamp\_Massey}(den, p, z) \in \mathbb{Z}_p[Z]$

27: $d \leftarrow \deg(\Lambda_{den}(Z))$

28: Let $roots_{den}$ be the distinct roots of $\Lambda_{den}(Z) \in \mathbb{Z}_p[Z]$.

29: **end if**

30: **if** $s = |roots_{num}|$ and $s < T$ **then** $numerator\_done \leftarrow$ true **end if**

31: **if** $d = |roots_{den}|$ and $d < T$ **then** $denominator\_done \leftarrow$ true **end if**

32: **if** $numerator\_done \wedge denominator\_done$ **then break end if**

33: $T\_old \leftarrow 2T, \qquad T \leftarrow 2T$

34: **end while** ▷ Recover monomials from roots using trial division

35: $\mathcal{M}_{num} \leftarrow \text{get\_monomial}(roots_{num}, Primes, n, vars)$

36: $\mathcal{M}_{den} \leftarrow \text{get\_monomial}(roots_{den}, Primes, n, vars)$

37: **if** $\mathcal{M}_{num} =$ FAIL or $\mathcal{M}_{den} =$ FAIL **then return** FAIL

38: **end if** ▷ Recover coefficients via Zippel Vandermonde solver

39: $A \leftarrow \text{Zippel\_Vandermonde\_solver}(num, s, roots_{num}, \Lambda_{num}(z), p)$

40: $B \leftarrow \text{Zippel\_Vandermonde\_solver}(den, d, roots_{den}, \Lambda_{den}(z), p)$

41: $f \leftarrow \sum_{m=1}^{s} A_m \mathcal{M}_{num} m, \qquad g \leftarrow \sum_{m=1}^{d} B_m \mathcal{M}_{den} m$

42: Let μ be the leading coefficient of gg in grlex order where $x_1 > \cdots > x_n$.

43: $f \leftarrow \mu^{-1} f \bmod \text{p}, \qquad g \leftarrow \mu^{-1} g \bmod \text{p}.$

44: **return** f, g.

Numerator Denominator Separation Algorithm (NDSA)

1: **Input:** Modular black box $\mathcal{B}$ for rational function $\frac{f(x_1,\ldots,x_n)}{g(x_1,\ldots,x_n)}$ over $\mathbb{Z}_p$, prime p, where $f, g \in K[x_1, \ldots, x_n]$, $\gcd(f, g) = 1$, $\sigma \in \mathbb{Z}_p^n$, $\beta \in \mathbb{Z}_p^{n-1}$, $num\_points \in \mathbb{N}$.

2: **Output:**

$$\hat{f}(x) = \frac{f(x, \beta_2(x - \sigma_1) + \sigma_2, \ldots, \beta_n(x - \sigma_1) + \sigma_n)}{c}$$

$$\hat{g}(x) = \frac{g(x, \beta_2(x - \sigma_1) + \sigma_2, \ldots, \beta_n(x - \sigma_1) + \sigma_n)}{c},$$

for some scalar $c \in \mathbb{Z}_p$.

3: $t \leftarrow num\_points$

4: **while** true **do**

5: Pick random points $\alpha = [\alpha_1, \ldots, \alpha_t] \in \mathbb{Z}_p^t$

6: $m(x) \leftarrow \prod_{k=1}^{t}(x - \alpha_k) \in \mathbb{Z}_p[x]$

7: $\mathcal{L} \leftarrow [\Psi_{\sigma,\beta}(\alpha_1), \ldots, \Psi_{\sigma,\beta}(\alpha_t)] \in \mathbb{Z}_p^{t\times n}$ such that: $\Psi_{\sigma,\beta}(\alpha_k) \leftarrow [\alpha_k, \beta_2(\alpha_k - \sigma_1) + \sigma_2, \ldots, \beta_n(\alpha_k - \sigma_1) + \sigma_n] \mod p \ \forall\ 1 \le k \le t$

8: $Y \leftarrow [y_1, \ldots, y_t] \in \mathbb{Z}_p^t$, where $y_i = \mathcal{B}(\mathcal{L}(\alpha_i), p)\ 1 \le i \le t$

9: **if** Y=FAIL **then** ▷ $g(\alpha_k) = 0$ for some $1 \le k \le t$, resample α

10: next

11: **end if**

12: $u(x) \leftarrow \text{Interpolate}(\alpha, Y, x) \mod p$

13: $result \leftarrow \text{MQRFR}(m, u) \mod p$

14: **if** result = FAIL **then** ▷ either $q \le 1$ or $g = 0$ or $\gcd(\hat{f}_i, \hat{g}_i) \ne 1$

15: **return** FAIL

16: **else**

17: $\hat{f}(x), \hat{g}(x), deg\_q \leftarrow result$

18: **end if**

19: **if** $deg\_q > 1$ **then**

20: **break**

21: **else**

22: $t \leftarrow 2t$

23: **end if**

24: **end while**

25: **return** $\hat{f}, \hat{g}$

4.6 Example of Kaltofen Yang multivariate rational function interpolation

Let $\mathcal{B} : \mathbb{Z}^2, p \longrightarrow \mathbb{Z}_p$ be the black box for the rational function $h(x,y) = \frac{x+5y}{xy+3} \in \mathbb{Z}_p(x,y)$ in $lex(x > y)$ that we wish to recover using the Kaltofen-Yang algorithm.
Suppose that we pick the prime $p = 107$.
The first step is to pick the parameters β and σ that define the affine line $\mathcal{L}_i$ using the regular morphism

$$\begin{array}{rccc} \Psi_{\sigma^i,\beta}: & \mathbb{Z}_{107}^1 & \longrightarrow & \mathbb{Z}_{107}^2 \\ & x & \longmapsto & (x, \beta(x-\sigma_1^i)+\sigma_2^i) \end{array}$$

$$\text{Let } \sigma^i = \begin{bmatrix} \sigma_1^i \\ \sigma_2^i \end{bmatrix} = \begin{bmatrix} 2^i \\ 3^i \end{bmatrix} \quad \text{and } \beta = 7$$

Substituting β and σ in equation 4.4,

$$\mathcal{L}_i := \mathrm{Im}(\Psi_{\sigma^i,\beta}(x)) := \begin{bmatrix} 1 \\ 7 \end{bmatrix} x + \begin{bmatrix} 0 \\ 3^i - 7 \cdot 2^i \end{bmatrix}. \tag{4.7}$$

Now, we restrict $\Psi_{\sigma^i,\beta}(x)$ to $\mathcal{L}_i := \mathrm{Im}(\Psi_{\sigma^i,\beta}(x)) \subset \mathbb{Z}_{107}^2$

$$\overline{\Psi}_{\sigma^i,\beta} : \mathbb{Z}_{107}^1 \longrightarrow \mathcal{L}_i \subset \mathbb{Z}_{107}^2 \tag{4.8}$$

Let t be the number of points that we want to sample on the affine line $\mathcal{L}_i$, here we take $t = 5$.

$$\text{Let } \alpha_j := j \in \mathbb{Z}_{107}, \text{ where } 1 \leq j \leq 5.$$

For $i = 0$ we get the first affine line $\mathcal{L}_0$

$$\mathcal{L}_0(x) := \begin{bmatrix} 1 \\ 7 \end{bmatrix} x + \begin{bmatrix} 0 \\ 1-7 \end{bmatrix} = \begin{bmatrix} 1 \\ 7 \end{bmatrix} x + \begin{bmatrix} 0 \\ -6 \end{bmatrix}.$$

Now we need to find the coordinates of the point $\mathcal{L}_0(\alpha_1)$ in the line $\mathcal{L}_0$ at $x = \alpha_1 = 1$.

$$\mathcal{L}_0(\alpha_1) := \begin{bmatrix} 1 \\ 7 \end{bmatrix} 1 + \begin{bmatrix} 0 \\ -6 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}.$$

We probe the modular black box $\mathcal{B}$ of the rational function $h = \dfrac{f}{g}$ at some point $\mathcal{L}_i(\alpha_j)$ on the affine line $\mathcal{L}_i$ for multiple distinct $\alpha_j \in \mathbb{Z}_p$, this is equivalent to restricting the rational function on the affine line $\mathcal{L}_i$. Geometrically, this is equivalent to composing $\frac{f}{g}$ with the

surjective regular morphism that defines the affine line $\overline{\Psi}_{\sigma^i,\beta}$

$$\frac{f}{g}\Big|_{\mathcal{L}_i} = \frac{f}{g} \circ \overline{\Psi}_{\sigma^i,\beta}(x) = \frac{f(\overline{\Psi}_{\sigma^i,\beta}(x))}{g(\overline{\Psi}_{\sigma^i,\beta}(x))} = T(x) \in \mathbb{Z}_p(x)$$

where $T(x)$ is a univariate rational function. Computationally, it looks like

$$\implies T_i(x) = \mathcal{B}([x, 7(x-\sigma_1^i)+\sigma_2^i], 107) = \frac{f(x, 7(x-\sigma_1^i)+\sigma_2^i)}{g(x, 7(x-\sigma_1^i)+\sigma_2^i)} \bmod 107 \tag{4.9}$$

For line $\mathcal{L}_0(x)$ we will have

$$T_0(x) = \mathcal{B}([x, 7(x-1)+1], 107) = \frac{f(x, 7(x-1)+1)}{g(x, 7(x-1)+1)} \bmod 107$$

For $\mathcal{L}_0(\alpha_1 = 1)$ we will have $T_0(1) = \mathcal{B}([1,1], 107) = \dfrac{f(1,1)}{g(1,1)} \bmod 107$
Now, we find $T_0(\alpha_j) =$ where $1 \leq j \leq 5$ by evaluating the black box $\mathcal{B}$ at $\mathcal{L}_0(\alpha_j)$.

$$T_0(\alpha_j) = \mathcal{B}([\alpha_j, 7(\alpha_j-1)+1], 107) = \frac{f(\alpha_j, 7(\alpha_j-1)+1)}{g(\alpha_j, 7(\alpha_j-1)+1)} \bmod 107$$

Now we can use the Maximal quotient rational function reconstruction algorithm to recover

Table 4.2: Computed values of $\overline{\Psi}_{\sigma^0,\beta}(\alpha_j) = \mathcal{L}_0(\alpha_j)$ and $T_0(\alpha_j)$

α_j	$\overline{\Psi}_{\sigma^0,\beta}(\alpha_j) = \mathcal{L}_0(\alpha_j)$	$T_0(\alpha_j) = \mathcal{B}(\alpha_j, 7(\alpha_j-1)+1)$
1	[1, 1]	55
2	[2, 8]	36
3	[3, 15]	15
4	[4, 22]	33
5	[5, 29]	95

the univariate image of the numerator and denominator separately. The modulus polynomial is $\overline{m}(x) = (x-1)(x-2)(x-3)(x-4)(x-5) = x^5 + 92x^4 + 85x^3 + 96x^2 + 60x + 94$.
The interpolating polynomial is $u_0(x) = 52x^4 + 4x^3 + 66x^2 + 45x + 102$.
Applying the maximal quotient rational function reconstruction algorithm (MQRFR) to $\overline{m}(x)$ and $u_0(x)$ modulo 107 yields the following sequence of quotients q_k, remainders r_k, and auxiliary polynomials t_k:

At the step where $\deg(q_i)$ is maximal ($i = 3$), we identify

$$\hat{f}_0(x) = 51x + 11, \qquad \hat{g}_0(x) = x^2 + 45x + 31.$$

Table 4.3: MQRFR computations for input polynomials $\overline{m}(x)$ and $u_0(x)$

k	q_k	r_k	t_k
0	–	$x^5 + 92x^4 + 85x^3 + 96x^2 + 60x + 94$	0
1	$35x + 32$	$52x^4 + 4x^3 + 66x^2 + 45x + 102$	1
2	$52x + 21$	$x^3 + 47x^2 + 79x + 40$	$72x + 75$
3	$21x^2 + 95x + 44$	$51x + 11$	$x^2 + 45x + 31$
4	–	91	$86x^4 + 30x^3 + 59x^2 + 69x + 102$

These are the univariate images of the rational multivariate function we are trying to recover.

$$T_0(x) = \frac{51x + 11}{x^2 + 45x + 31}$$

Here, $T_0(x)$ is the univariate restriction of the rational function $\dfrac{f}{g}$ to the line $\mathcal{L}_0$ that we recovered from points $\mathcal{L}_0(\alpha_j)$ where $1 \leq j \leq 5$.
For this example, we will reuse the alpha values, so our modulus polynomial $\overline{m}$ is the same for each iteration, only the interpolating polynomial u changes. For $i = 1$, we have $\sigma^1 = \begin{bmatrix} 2 \\ 3 \end{bmatrix}$, our affine line $\mathcal{L}_1$

$$\mathcal{L}_1 := \begin{bmatrix} 0 \\ 3 - 7 \cdot 2 \end{bmatrix} + x \begin{bmatrix} 1 \\ 7 \end{bmatrix} = \begin{bmatrix} 0 \\ -11 \end{bmatrix} + x \begin{bmatrix} 1 \\ 7 \end{bmatrix}.$$

Now we need to find the coordinates of the point $\mathcal{L}_0(\alpha_1)$ in the line $\mathcal{L}_1$ at $x = \alpha_1 = 1$.

$$\mathcal{L}_1(\alpha_1) := \begin{bmatrix} 0 \\ -11 \end{bmatrix} + 1 \begin{bmatrix} 1 \\ 7 \end{bmatrix} = \begin{bmatrix} 1 \\ -4 \end{bmatrix} = \begin{bmatrix} 1 \\ 103 \end{bmatrix} \mod 107.$$

Computationally, we will have,

$$T_1(x) = \mathcal{B}([x, 7(x-2)+3], 107) = \frac{f(x, 7(x-2)+3)}{g(x, 7(x-2)+3)} \mod 107$$

Table 4.4: Computed values of $\overline{\Psi}_{\sigma^1,\beta}(\alpha_j) = \mathcal{L}_1(\alpha_j)$ and $T_1(\alpha_j)$

α_j	$\overline{\Psi}_{\sigma^1,\beta}(\alpha_j) = \mathcal{L}_1(\alpha_j)$	$T_1(\alpha_j) = \mathcal{B}([\alpha_j, 7(\alpha_j - 1) + 1], 107)$
1	[1, 103]	19
2	[2, 3]	97
3	[3, 10]	47
4	[4, 17]	54
5	[5, 24]	68

Table 4.5 presents the remaining output. Where $\overline{\Psi}_{\sigma^i,\beta}(\alpha_j) = [\alpha_j, \beta(\alpha_j - \sigma_1^i) + \sigma_2^i]$ and $Y_i = \mathcal{B}(\overline{\Psi}_{\sigma^i,\beta}(\alpha_j), 107)$.

Table 4.5: Values of σ_i, $\overline{\Psi}_{\sigma^i,\beta}(\alpha_j) = \mathcal{L}_i(\alpha_j)$,$Y_i$, u_i, $\hat{f}_i(x), \hat{g}_i(x)$ and related quantities.

i	σ^i	$\mathcal{L}_i(\alpha_j)$	Y_i	$u_i(x)$	$\hat{f}_i(x)$	$\hat{g}_i(x)$
0	$\begin{bmatrix}1\\1\end{bmatrix}$	$\begin{bmatrix}[1,1]\\[2,8]\\[3,15]\\[4,22]\\[5,29]\end{bmatrix}$	$\begin{bmatrix}55\\36\\15\\33\\95\end{bmatrix}$	$52x^4 + 4x^3 + 66x^2 + 45x + 102$	$51x + 11$	$x^2 + 45x + 31$
1	$\begin{bmatrix}2\\3\end{bmatrix}$	$\begin{bmatrix}[1,103]\\[2,3]\\[3,10]\\[4,17]\\[5,24]\end{bmatrix}$	$\begin{bmatrix}19\\97\\47\\54\\68\end{bmatrix}$	$66x^4 + 102x^3 + 28x^2 + 2x + 35$	$51x + 38$	$x^2 + 29x + 31$
2	$\begin{bmatrix}4\\9\end{bmatrix}$	$\begin{bmatrix}[1,95]\\[2,102]\\[3,2]\\[4,9]\\[5,16]\end{bmatrix}$	$\begin{bmatrix}66\\95\\49\\4\\99\end{bmatrix}$	$16x^4 + 31x^3 + 72x^2 + 105x + 56$	$51x + 17$	$x^2 + 89x + 31$
3	$\begin{bmatrix}8\\27\end{bmatrix}$	$\begin{bmatrix}[1,85]\\[2,92]\\[3,99]\\[4,106]\\[5,6]\end{bmatrix}$	$\begin{bmatrix}17\\78\\68\\1\\27\end{bmatrix}$	$77x^4 + 17x^3 + 24x^2 + 106x + 7$	$51x + 71$	$x^2 + 57x + 31$
4	$\begin{bmatrix}16\\81\end{bmatrix}$	$\begin{bmatrix}[1,83]\\[2,90]\\[3,97]\\[4,104]\\[5,4]\end{bmatrix}$	$\begin{bmatrix}77\\51\\81\\25\\29\end{bmatrix}$	$12x^4 + 106x^3 + 55x^2 + 64x + 54$	$51x + 39$	$x^2 + 72x + 31$
5	$\begin{bmatrix}32\\29\end{bmatrix}$	$\begin{bmatrix}[1,26]\\[2,33]\\[3,40]\\[4,47]\\[5,54]\end{bmatrix}$	$\begin{bmatrix}82\\66\\6\\78\\50\end{bmatrix}$	$90x^4 + 21x^3 + 63x^2 + 10x + 5$	$51x + 90$	$x^2 + 18x + 31$
6	$\begin{bmatrix}64\\87\end{bmatrix}$	$\begin{bmatrix}[1,74]\\[2,81]\\[3,88]\\[4,95]\\[5,102]\end{bmatrix}$	$\begin{bmatrix}34\\31\\77\\6\\69\end{bmatrix}$	$4x^4 + 75x^3 + 63x^2 + 79x + 27$	$51x + 2$	$x^2 + 86x + 31$
7	$\begin{bmatrix}21\\47\end{bmatrix}$	$\begin{bmatrix}[1,14]\\[2,21]\\[3,28]\\[4,35]\\[5,42]\end{bmatrix}$	$\begin{bmatrix}86\\0\\41\\2\\106\end{bmatrix}$	$9x^4 + 36x^3 + 104x^2 + 71x + 80$	$51x + 5$	$x^2 + x + 31$

We evaluate the recovered univariate numerators and denominators at different $\sigma_1^i = 2^i$ to generate the independent linear recurrence sequences for the numerator and the denominator, respectively.

$$\hat{f}_i(\sigma_1^i) := [62, 33, 7, 51, 106, 10, 56, 6]$$

$$\hat{g}_i(\sigma_1^i) := [77, 93, 82, 16, 48, 26, 1, 65]$$

We use the Berlekamp Massey Euclidean algorithm to find the minimal characteristic polynomial for the sequence corresponding to the numerator and the denominator.

Numerator

We compute the syndrome polynomial defined in 3.3.1 using the terms of the sequence

$$S_N(Z) = 62Z^7 + 33Z^6 + 7Z^5 + 51Z^4 + 106Z^3 + 10Z^2 + 56Z + 6$$

Table 4.6: Berlekamp–Massey algorithm over $\mathbb{Z}_{107}$ for the sequence $s_i = f(2^i, 3^i)$.

i	r_i	t_i
0	Z^8	0
1	$62Z^7 + 33Z^6 + 7Z^5 + 51Z^4 + 106Z^3 + 10Z^2 + 56Z + 6$	1
2	$92Z^6 + 32Z^5 + 36Z^4 + 95Z^3 + 45Z^2 + 83Z + 2$	$88Z + 36$
3	$Z + 19$	$57Z^2 + 36Z + 21$

$$\Lambda_N(Z) = 57Z^2 + 36Z + 21$$

Making $\Lambda_N(Z)$ monic by normalizing it gives us

$$\Lambda_N(Z) = Z^2 + 102Z + 6$$

We now find the monomials in the numerator by factorizing the minimal characteristic polynomial for the numerator using maple's Cantor-Zassenhaus algorithm

$$\Lambda_N(Z) = (Z - 2)(Z - 3)$$

We recover the monomials for the numerator by factorizing the roots of characteristic polynomial for the numerator

$$2 \longleftrightarrow x$$

$$3 \longleftrightarrow y$$

$$f(x,y) = \begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \end{bmatrix}$$

We use Zippel's transpose Vandermonde Solver to find the coefficients of the numerator f

$$\begin{bmatrix} 1 & 1 \\ 2 & 3 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \end{bmatrix} = \begin{bmatrix} 62 \\ 33 \end{bmatrix} \pmod{107}.$$

Solving this system using Zippel's Vandermonde matrix transpose over $\mathbb{Z}_{107}$ gives us the coefficients $c_0 = 46$ and $c_1 = 16$.

Denominator

The syndrome polynomial corresponding to the denominator evaluations is

$$S_D(Z) = 77Z^7 + 93Z^6 + 82Z^5 + 16Z^4 + 48Z^3 + 26Z^2 + Z + 65 \in \mathbb{Z}_{107}[Z].$$

Table 4.7: Berlekamp–Massey algorithm over $\mathbb{Z}_{107}$for the sequence $s_i = g(2^i, 3^i)$.

i	r_i	t_i
0	Z^8	0
1	$77Z^7 + 93Z^6 + 82Z^5 + 16Z^4 + 48Z^3 + 26Z^2 + Z + 65$	1
2	$2Z^6 + 14Z^5 + 86Z^4 + 90Z^3 + 7Z^2 + 44Z + 62$	$25Z + 24$
3	$43Z + 88$	$54Z^2 + 50Z + 3$

$$\Lambda_D(Z) = 54Z^2 + 50Z + 3$$

Making $\Lambda_D(Z)$ monic by normalizing it gives us

$$\Lambda_D(Z) = Z^2 + 100Z + 6$$

We now find the monomials in the denominator by factorizing the minimal characteristic polynomials for the denominator using Maple's Cantor-Zassenhaus algorithm.

$$\Lambda_D(Z) = (Z - 1)(Z - 6)$$

We recover the monomials for the denominator by factorizing the roots of characteristic polynomial for the denominator

$$2 \longleftrightarrow x$$

$$3 \longleftrightarrow y$$

$$1 = 2^0 3^0 \longleftrightarrow 1, \qquad 6 = 2 \cdot 3 \longleftrightarrow xy$$

$$g(x, y) = \begin{bmatrix} xy & 1 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \end{bmatrix}$$

We use Zippel's transpose Vandermonde Solver to find the coefficients of the denominator g

$$\begin{bmatrix} 1 & 1 \\ 1 & 6 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \end{bmatrix} = \begin{bmatrix} 77 \\ 93 \end{bmatrix} \pmod{107}.$$

Solving this system using Zippel's Vandermonde matrix transpose over $\mathbb{Z}_{107}$ gives us the coefficients $c_0 = 46$ and $c_1 = 31$.
Normalizing to make the denominator monic gives us

$$g(x, y) = \frac{46xy + 31}{46} \bmod 107 = xy + 3$$

Divding the numerator by the leading coefficient as well gives us,

$$f(x, y) = \frac{46x + 16y}{46} \bmod 107 = x + 5y$$

The final multivariate rational function is,

$$\frac{f(x, y)}{g(x, y)} = \frac{x + 5y}{xy + 3}$$

Chapter 5

Solving Parametric Linear Systems using Sparse Multivariate Rational Function Interpolation

5.1 Introduction

Consider the parametric linear system $Ax = b$ where

$$A \in \mathbb{Z}[y_1, y_2, \ldots, y_m]^{n \times n} \quad \text{and } b \in \mathbb{Z}[y_1, y_2, \ldots, y_m]^n.$$

Assuming rank A= n, the solution vector $x \in \mathbb{Z}(y_1, \ldots, y_m)^n$, that is,

$$x = \begin{bmatrix} x_1 & x_2 & \cdots & x_n \end{bmatrix}^T = \begin{bmatrix} \frac{f_1}{g_1} & \frac{f_2}{g_2} & \cdots & \frac{f_n}{g_n} \end{bmatrix}^T$$

for some $f_i, g_i \in \mathbb{Q}[y_1, y_2, \ldots, y_m]$, $g_i \neq 0$, $g_i \mid \det(A)$, and $\gcd(f_i, g_i) = 1$ for $1 \leq i \leq n$ and leading coefficient of $g_i = 1$ in some term ordering.

Cramer's rule

From Cramer's rule, we know that the solutions of $Ax = b$ are given by

$$x_i = \frac{\det(A_i)}{\det(A)} \in \mathbb{Z}(y_1, \ldots, y_m) \implies \det(A_i) = x_i \det(A)$$

where A_i is the matrix obtained by replacing the i-th column of A with b, and $\det(A), \det(A_i) \in \mathbb{Z}[y_1, \ldots, y_m]$. To simplify the fraction $\dfrac{\det(A_i)}{\det(A)}$ we compute $h_i = \gcd(\det(A), \det(A_i))$ and set

$$x_i = \frac{\det(A_i)/h_i}{\det(A)/h_i}$$

We remark that the final solution for x_i may be smaller than the $\frac{\det(A_i)}{\det(A)}$, that is $\#\frac{\det(A_i)}{h_i} \ll \det(A_i)$ and $\#\frac{\det(A)}{h_i} \ll \det(A_i)$ is possible.

The Bareiss/Edmonds/Lipson algorithm

The Bareiss/Edmonds/Lipson fraction-free Gaussian elimination algorithm [1, 8, 15], triangularizes an augmented matrix $B = [A \mid b]$ to obtain $\det(A) \in \mathbb{Z}[y_1, y_2, \ldots, y_m]$, and then the Lipson algorithm computes $\det(A_i)$ in the ring $\mathbb{Z}[y_1, y_2, \ldots, y_m]$ by back substitution.

Expression swell in Bareiss/Edmonds/Lipson algorithm

The Bareiss/Edmonds/Lipson algorithm computes the leading principal minors $B_{k,k}$(determinant of the principal $k \times k$ submatrix of A) . However, in the last step the determinant of coefficient matrix A is computed using

$$B_{n,n} = \frac{B_{n-1,n-1}B_{n,n} \; - \; B_{n,n-1}B_{n-1,n}}{B_{n-2,n-2}} = \det(A) \tag{5.1}$$

so we divide $N = B_{n-1,n-1}B_{n,n} \; - \; B_{n,n-1}B_{n-1,n}$ by $B_{n-2,n-2}$ to get $\det(A)$ .The numerator N in equation 5.1 is the product of the determinants $B_{n,n}$ and $B_{n-2,n-2}$, can be much larger than $\det(A)$.

Geometric interpretation of parametric linear system of equations

Let $A\mathbf{x} = b$ be a linear system such that $A_{n\times n}$ and $\forall\ a_{ij} \in A$, $a_{ij} \in \mathbb{F}[y_1, \ldots, y_m]$. Let the solution of the system be,

$$x_i = \frac{f_i(y_1, \ldots, y_m)}{g_i(y_1, \ldots, y_m)}$$

The solution vectors define the coordinate functions of a rational map,

$$\begin{array}{rcl} \Phi : \mathbb{F}^m & \dashrightarrow & \mathbb{F}^n \\ (a_1, \ldots, a_m) & \longmapsto & \left(\frac{f_1(a_1, \ldots, a_m)}{g_1(a_1, \ldots, a_m)}, \ldots, \frac{f_n(a_1, \ldots, a_m)}{g_n(a_1, \ldots, a_m)}\right) \end{array} \tag{5.2}$$

5.2 Example of solving a parametric linear system using Kaltofen Yang multivariate rational function interpolation

Consider the following 2×2 system of linear equations with parameters y_1 and y_2 that we wish to solve.

$$y_1x_1 + y_1x_2 = 1 \tag{5.3}$$

$$y_1y_2x_1 - x_2 = 1 \tag{5.4}$$

Writing it as a linear system $A\mathbf{x} = b$ gets us,

$$A := \begin{bmatrix} y_1 & y_1 \\ y_1 y_2 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

The coefficient matrix A

$$A := \begin{bmatrix} y_1 & y_1 \\ y_1 y_2 & -1 \end{bmatrix} \text{ contains entries in } \mathbb{Z}[y_1, y_2].$$

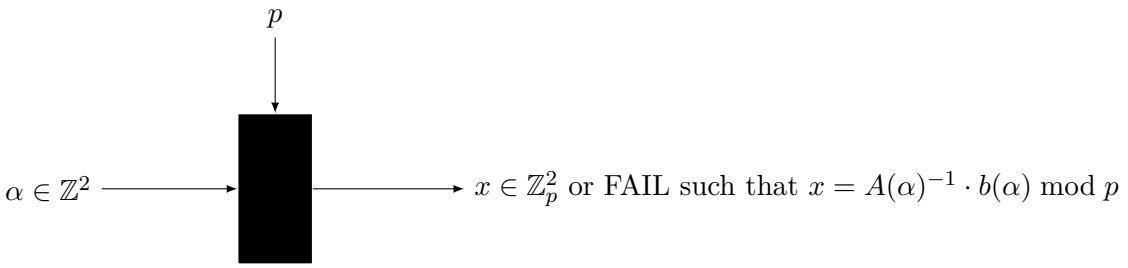
The augmented matrix for the system is given by,

$$\left[\begin{array}{cc|c} y_1 & y_1 & 1 \\ y_1 y_2 & -1 & 1 \end{array}\right]$$

The solution of this system $\mathbf{x} \in \mathbb{Z}(y_1, y_2)^2$ is a vector of multivariate rational functions with components $x_1, x_2 \in \mathbb{Z}(y_1, y_2)$.

$$x_1 = \frac{y_1 + 1}{y_1^2 y_2 + y_1}, \quad x_2 = \frac{y_2 - 1}{y_1 y_2 + 1}. \tag{5.5}$$

The black box for the parametric linear system can be visualized as,



It is implemented in Maple as follows,

```
# Assumptions:
#   A      : n x n Matrix with entries in F[params]
#   b      : length-n Vector with entries in F[params]
#   Vars   : [x1, ..., xn]        (unknowns)
#   params : [y1, ..., ym]        (parameters)

B := proc(alpha::list(integer), p::prime)
    local n, L_alpha, A_alpha, b_alpha, x,m;
    uses LinearAlgebra:-Modular:
    global counter;     counter := counter + 1:
    m:=numelems(params);
    # L_alpha = y_i \gets \alpha_i,  i = 1,...,m
    L_alpha := { seq(params[i] = alpha[i], i = 1 .. m) }:
    # A(alpha) and b(alpha) over F_p
```

```
    A_alpha := Eval(A, L_alpha) mod p:
    b_alpha := Eval(b, L_alpha) mod p:

    # Solve A(alpha) x = b(alpha) in F_p^n
    x := traperror(LinearSolve(p, A_alpha, b_alpha)):
    if x = "matrix is singular" then
        return FAIL
    fi:
    return convert(x, list):
end:
```

To demonstrate the practical execution of the Kaltofen-Yang algorithm, we consider the problem of solving a parametric linear system $A\mathbf{x} = \mathbf{b}$, where the entries of the matrix A are polynomials in $\mathbb{Z}[y_1, y_2]$. The solution vector $\mathbf{x}$ consists of multivariate rational functions $x_1, x_2 \in \mathbb{Q}(y_1, y_2)$ that we treat as black box objects. In this example, our goal is to reconstruct the rational functions x_1 and x_2 by probing the black box along parallel affine lines $\mathcal{L}_i$ defined by $\overline{\Psi}_{\sigma^i,\beta}$. We effectively slice the 2-dimensional parameter space (with coordinates y_1, y_2) using the lines $\mathcal{L}_i$.
By restricting the system to these lines, one line at at time, the multivariate solution vector $\mathbf{x}(y_1, y_2)$ collapses into a vector of univariate rational functions

$$T_i(y) = (T_{i1}(y), T_{i2}(y)) \in \mathbb{Z}_p(y)^2, \quad \text{where} \quad T_{i1}(y) = \frac{\hat{f}_{i1}(y)}{\hat{g}_{i1}(y)}, \qquad T_{i2}(y) = \frac{\hat{f}_{i2}(y)}{\hat{g}_{i2}(y)}.$$

The algorithm proceeds by solving the numerical linear system at several discrete points $\overline{\Psi}_{\sigma^i,\beta}(\alpha_j)$ along the lines $\mathcal{L}_i$. These numerical solutions are then aggregated and given as input to the MQRFR algorithm to implicitly reconstruct the univariate functions $T_{i1}(y)$ and $T_{i2}(y)$ by recovering the univariate numerators and the denominators,

$$\mu_1\hat{f}_{i1}(y), \mu_2\hat{f}_{i2}(y), \mu_1\hat{g}_{i1}(y), \mu_1\hat{g}_{i2}(y) \in \mathbb{Z}_p[y], \text{ where } \mu_1 = \text{lcoeff}(\hat{g}_{i1}(x)), \mu_2 = \text{lcoeff}(\hat{g}_{i2}(x)) \in \mathbb{Z}_p.$$

Our construction of $\overline{\Psi}_{\sigma^i,\beta}$ from Kaltofen and Yang's ring isomorphism ϕ_1, ensures that the following property holds true,

$$\mu_1\hat{f}_{i1}(\sigma_1^i) = \mu_1 f_1(\sigma^i), \qquad \mu_2\hat{f}_{i2}(\sigma_1^i) = \mu_2 f_2(\sigma^i), \qquad \mu_1\hat{g}_{i1}(\sigma_1^i) = \mu_1 g_1(\sigma^i), \qquad \mu_2\hat{g}_{i2}(\sigma_1^i) = \mu_2 g_2(\sigma^i).$$

This allows us to treat the values $\{\mu_1\hat{f}_{i1}(\sigma_1^i)\}, \{\mu_2\hat{f}_{i2}(\sigma_1^i)\}$, $\{\mu_1\hat{g}_{i1}(\sigma_1^i)\}$, and $\{\mu_2\hat{g}_{i2}(\sigma_1^i)\}$ as if they came from probing the black boxes of the multivariate numerators $\mu_1 f_1, \mu_2 f_2 \in \mathbb{Z}_p[y_1, y_2]$ and denominators $\mu_1 g_1, \mu_2 g_2 \in \mathbb{Z}_p[y_1, y_2]$ at the sequence of points $\sigma^1, \sigma^2, \ldots, \sigma^i$. We then construct the syndrome sequences for the Berlekamp-Massey algorithm by evalu-

ating the recovered univariate numerators and denominators at σ_1^i,

$$S_{f1} = \{\mu_1 \hat{f}_{i1}(\sigma_1^i)\}_{i\geq 0}, \quad S_{f2} = \{\mu_2 \hat{f}_{i2}(\sigma_1^i)\}_{i\geq 0}, \quad S_{g1}\{\mu_1 \hat{g}_{i1}(\sigma_1^i)\}_{i\geq 0}, \quad S_{g2} = \{\mu_2 \hat{g}_{i2}(\sigma_1^i)\}_{i\geq 0}.$$

We compute and factorize the corresponding minimal characteristic polynomials $\Lambda_{fi}(Z), \Lambda_{gi}(Z)$, to get the roots $\{r_{\Lambda_{f1}}\}, \{r_{\Lambda_{f2}}\}, \{r_{\Lambda_{g1}}\}, \{r_{\Lambda_{g2}}\}$. We recover the monomials in $f_1, f_2, g_1, g_2 \in \mathbb{Z}_p[y_1, y_2]$ by performing trial division of the roots of the minimal characteristic polynomial by $\sigma = [2, 3]$.
To get the coefficients of $f_1.f_2, g_1, g_2$, we solve the transposed Vandermonde systems using Zippel's Vandermonde Solver and reconstruct the final solution.

$$x_1 = \frac{y_1 + 1}{y_1^2 y_2 + y_1}, \quad x_2 = \frac{y_2 - 1}{y_1 y_2 + 1}.$$

Our algorithm took 54 black box probes to recover $\frac{f_1}{g_1}, \frac{f_2}{g_2}$.
There is also a Maple demo for solving this system using the algorithm here in section 5.4.

5.3 Rational B-spline system

A rational B-spline system is used to characterize curves using piecewise multivariate rational functions in computer graphics. Rational B-spline [18] generalizes standard B-splines by adding weights to control points, allowing it to represent both free-form shapes and exact analytical shapes like circles, ellipses, and lines, making it the industry standard (NURBS) in CAD/CAM, graphics, and engineering for precise, flexible modeling.

5.3.1 Example Rational B-spline system

Parametric linear systems often show up naturally in rational B-splines. Consider the parameterized system of linear equations from a rational B-spline system in 5.6 If we write the system as $A\mathbf{x} = b$, then $A \in \mathbb{Z}[y_1, \ldots, y_5]^{21\times 21}$ The solution vector $\mathbf{x} \in \mathbb{Z}(y_1, \ldots, y_5)^{21}$ and vector $b \in \mathbb{Z}^{21}$.
Let $\mathbf{x} = [x_1, x_2, \ldots, x_{21}]^T$ be the solution of this system, then

$$x_i = \frac{f_i(y_1, \ldots, y_5)}{g_i(y_1, \ldots, y_5)} \in \mathbb{Z}(y_1 \ldots, y_5) \quad \text{where } 1 \leq i \leq 21.$$

$$
\begin{aligned}
x_7 + x_{12} &= 1 \\
x_8 + x_{13} &= 1 \\
x_{21} + x_6 + x_{11} &= 1 \\
x_1 y_1 + x_1 - x_2 &= 0 \\
x_{11} y_3 + x_{11} - x_{12} &= 0 \\
x_{16} y_5 - x_{17} y_5 - x_{17} &= 0 \\
-x_{20} y_3 + x_{21} y_3 + x_{21} &= 0 \\
x_3 y_2 + x_3 - x_4 &= 0 \\
-x_8 y_4 + x_9 y_3 + x_9 &= 0 \\
2x_1 y_1^2 - 2x_1 - 2x_{10} + 4x_2 &= 0 \\
-x_{10} y_2 + x_{18} y_2 + x_{18} - x_{19} &= 0 \\
2x_{11} y_3^2 - 2x_{11} + 4x_{12} - 2x_{13} &= 0 \\
-x_{13} y_4 + x_{14} y_4 + x_{14} - x_{15} &= 0 \\
2x_{15} y_5^2 - 4x_{16} y_5^2 + 2x_{17} y_5^2 - 2x_{17} &= 0 \\
2x_{19} y_3^2 - 4x_{20} y_3^2 + 2x_{21} y_3^2 - 2x_{21} &= 0 \\
2x_3 y_2^2 - 2x_3 + 4x_4 - 2x_5 &= 0 \\
-x_5 y_3 + x_6 y_3 + x_6 - x_7 &= 0 \\
2x_7 y_4^2 - 4x_8 y_4^2 + 2x_9 y_4^2 - 2x_9 &= 0 \\
-4x_{10} y_2^2 + 2x_{18} y_2^2 + 2x_2 y_2^2 - 2x_{18} + 4x_{19} - 2x_{20} &= 0 \\
2x_{12} y_4^2 - 4x_{13} y_4^2 + 2x_{14} y_4^2 - 2x_{14} + 4x_{15} - 2x_{16} &= 0 \\
2x_4 y_3^2 - 4x_5 y_3^2 + 2x_6 y_3^2 - 2x_6 + 4x_7 - 2x_8 &= 0
\end{aligned}
\tag{5.6}
$$

We present the solution of this parameterized linear system using the Kaltofen Yang.

$$
\begin{aligned}
x_1 &= \frac{1}{y_1^2 y_2 + y_1^2 + y_1 y_2 + 2y_1 + 1}, \\
x_2 &= \frac{1}{y_1 y_2 + y_1 + 1}, \\
x_3 &= \frac{1}{y_2^2 y_3 + y_2^2 + y_2 y_3 + 2y_2 + 1}, \\
x_4 &= \frac{1}{y_2 y_3 + y_2 + 1}, \\
x_5 &= \frac{y_2 + 1}{y_2 y_3 + y_2 + 1},
\end{aligned}
$$

$$x_6 = \frac{4y_2y_3^3y_4 - 2y_2y_3^2y_4^2 + 7y_2y_3^2y_4 - 2y_2y_3y_4^2 + 2y_3^3y_4 - y_3^2y_4^2 + 2y_2y_3^2 + 3y_2y_3y_4 + 5y_3^2y_4 - 2y_3y_4^2 + 2y_2y_3 + y_3^2 + 3y_3y_4 + 2y_3}{2y_2y_3^4y_4 - y_2y_3^3y_4^2 + 7y_2y_3^3y_4 - 3y_2y_3^2y_4^2 + y_2y_3^3 + 9y_2y_3^2y_4 + 3y_2y_3^2 - 3y_2y_3y_4^2 + 5y_2y_3y_4 + 3y_2y_3 - y_2y_4^2 + y_2y_4 + y_2 + 2y_3^3y_4 - y_3^2y_4^2 + 5y_3^2y_4 - 2y_3y_4^2 + y_3^2 + 4y_3y_4 + 2y_3 - y_4^2 + y_4 + 1},$$

$$x_7 = \frac{2y_3^2y_4 - y_3y_4^2 + 2y_3y_4 + y_3}{2y_3^2y_4 - y_3y_4^2 + 3y_3y_4 - y_4^2 + y_3 + y_4 + 1},$$

$$x_8 = \frac{y_4y_3}{2y_3y_4 - y_4^2 + y_4 + 1},$$

$$x_9 = \frac{y_3y_4^2}{2y_3^2y_4 - y_3y_4^2 + 3y_3y_4 - y_4^2 + y_3 + y_4 + 1},$$

$$x_{10} = \frac{y_1 + 1}{y_1y_2 + y_1 + 1},$$

$$x_{11} = \frac{y_3y_4 - y_4^2 + y_4 + 1}{2y_3^3y_4 - y_3^2y_4^2 + 5y_3^2y_4 - 2y_3y_4^2 + y_3^2 + 4y_3y_4 + 2y_3 - y_4^2 + y_4 + 1},$$

$$x_{12} = \frac{y_3y_4 - y_4^2 + y_4 + 1}{2y_3^2y_4 - y_3y_4^2 + 3y_3y_4 - y_4^2 + y_3 + y_4 + 1},$$

$$x_{13} = \frac{y_3y_4 - y_4^2 + y_4 + 1}{2y_3y_4 - y_4^2 + y_4 + 1},$$

$$x_{14} = \frac{2y_3^2y_4^3y_5 - 2y_3y_4^4y_5 + 2y_3^2y_4^3 + y_3^2y_4^2y_5 - 2y_3y_4^3 + 2y_3y_4^3y_5 - y_4^4y_5 + 2y_3^2y_4^2 + y_3y_4^3 + 4y_3y_4^2y_5 - y_4^4 + 6y_3y_4^2 + y_3y_4y_5 - y_4^3 + 2y_4^2y_5 + 2y_3y_4 + 3y_4^2 + y_4y_5 + 2y_4}{2y_3^2y_4^3y_5 - y_3y_4^4y_5 + 2y_3^2y_4^3 + 2y_3^2y_4^2y_5 - y_3y_4^3 + 2y_3y_4^3y_5 - y_4^4y_5 + 4y_3^2y_4^2 + y_3y_4^3 + 4y_3y_4^2y_5 - y_4^4 + 2y_3^2y_4 + 6y_3y_4^2 + y_3y_4y_5 - y_4^3 + 2y_4^2y_5 + 5y_3y_4 + 2y_4^2 + y_4y_5 + y_3 + 3y_4 + 1},$$

$$x_{15} = \frac{y_3^2y_4^3y_5 - y_3y_4^4y_5 + y_3^2y_4^2y_5 + 3y_3y_4^2y_5 - y_4^3y_5 + y_3y_4y_5 + y_4^2y_5 + y_4y_5 + y_3^2y_4^3 - y_3y_4^4 + y_3^2y_4^2 + 3y_3y_4^2 - y_4^3 + y_3y_4 + y_4^2 + y_4}{2y_3^2y_4^2y_5 - y_3y_4^3y_5 + 2y_3^2y_4^2 + 3y_3y_4^2y_5 - y_4^3y_5 + 2y_3^2y_4 + y_3y_4y_5 + y_4^2y_5 + y_3y_4 + y_4y_5 - y_3y_4^2 + 3y_3y_4 - y_4^2 + y_3 + y_4 + y_5 + 1},$$

$$x_{16} = \frac{y_3^2y_4^3y_5 - y_3y_4^4y_5 + y_3^2y_4^2y_5 + 3y_3y_4^2y_5 - y_4^3y_5 + y_3y_4y_5 + y_4^2y_5 + y_4y_5}{2y_3^2y_4^2y_5 - y_3y_4^3y_5 + 2y_3^2y_4^2 + 3y_3y_4^2y_5 - y_4^3y_5 + 2y_3^2y_4 + y_3y_4y_5 + y_4^2y_5 + y_3y_4 + y_4y_5 - y_3y_4^2 + 3y_3y_4 - y_4^2 + y_3 + y_4 + y_5 + 1},$$

$$x_{17} = \frac{y_3^2y_4^3y_5^2 - y_3y_4^4y_5^2 + y_3^2y_4^2y_5^2 + 3y_3y_4^2y_5^2 - y_4^3y_5^2 + y_3y_4y_5^2 + y_4^2y_5^2 + y_4y_5^2}{2y_3^2y_4^2y_5^2 - y_3y_4^3y_5^2 + 3y_3y_4^2y_5^2 - y_4^3y_5^2 + y_3y_4y_5^2 + y_4^2y_5^2 + y_4y_5^2 + 4y_3^2y_4^2y_5 - 2y_3y_4^3y_5 + 6y_3y_4^2y_5 - 2y_4^3y_5 + 2y_3^2y_4y_5 + 3y_3y_4y_5 + 3y_4^2y_5 + y_3y_5 + 2y_4y_5 + y_5 + 2y_3^2y_4^2 - y_3y_4^3 + 3y_3y_4^2 - y_4^3 + 2y_3^2y_4 + y_3y_4 + y_4^2 + y_3 + 2y_4 + 1},$$

$$x_{18} = \frac{2y_1y_2^2y_3 + 2y_1y_2^2 + y_1y_2y_3 + y_2^2y_3 + 2y_1y_2 + y_2^2 + y_2y_3 + 2y_2}{y_1y_2^3y_3 + y_1y_2^3 + 2y_1y_2^2y_3 + 3y_1y_2^2 + 2y_1y_2y_3 + 3y_1y_2 + y_1 + y_2^2y_3 + y_2^2 + y_2y_3 + 2y_2 + 1},$$

$$x_{19} = \frac{y_2y_3 + y_2}{y_2y_3 + y_2 + 1},$$

$$x_{20} = \frac{y_2y_3}{y_2y_3 + y_2 + 1},$$

$$x_{21} = \frac{y_2y_3^2}{y_2y_3^2 + 2y_2y_3 + y_2 + y_3 + 1}.$$

It took 973 black box probes to solve this system.

Table comparing expression swell

This very rational B-spline sytem 5.6 has been solved using the Bareiss/Edmonds/Lipson algorithm in [11]. We present some of the relevant metrics featured in [11].
For the solution of the rational B-spline computed using Kaltofen Yang algorithm, let 5.6, $x = x_i = \frac{f_i}{g_i}$ where $1 \leq i \leq 21$, let the number of terms in the numerator f_i be denoted as $\#f_i$ and in the denominator g_i be denoted as $\#g_i$. Let A be the coefficient matrix of the system 5.6 when it is written as $Ax = b$. Then $\#\det(A) = 1033$.
Let $\tilde{x}$ be the solution of the same rational B-spline system 5.6 computed using Bareiss/Edmonds/Lipton algorithm [11] , where $\tilde{x} = \tilde{x}_i = \frac{N_i}{D_i}$ for $1 \leq i \leq 21$. Let $\#N_i$ be the number of terms in the numerator N_i and $\#D_i$ be the number of terms in the denominator D_i .

We present a table in 5.1 listing the $\#f_i$, $\#g_i$, $\#N_i$ and $\#D_i$ where $B_{n-2,n-2}$ is the denominator in $B_{n,n} = \frac{B_{n-1,n-1}B_{n,n} - B_{n,n-1}B_{n-1,n}}{B_{n-2,n-2}}$.

Table 5.1: Number of terms in N_i, D_i, f_i and g_i.

	1	2	3	4	5	6	7	8	9	10	11
$\#N_i$	586	3,490	7,410	4,940	7,072	11,793	12,802	11,211	9,620	1,172	1,197
$\#D_i$	2	36	153	153	432	672	672	672	672	3	6
$\#f_i$	1	1	1	1	2	14	4	1	1	2	4
$\#g_i$	5	3	5	3	3	23	7	4	7	3	10

	12	13	14	15	16	17	18	19	20	21
$\#N_i$	1,827	2,142	1,666	2,072	1,320	1,320	2,650	2,543	3,971	5,675
$\#D_i$	9	9	9	9	9	18	18	27	36	117
$\#f_i$	4	4	19	16	8	8	8	2	1	1
$\#g_i$	7	4	22	16	16	26	12	3	3	5

The results demonstrate that the Kaltofen Yang sparse multivariate rational function interpolation effectively eliminates expression swell for the rational B-spline system 5.6. As shown in table 5.1, the number of terms is reduced by a factor of over 1,000 in the most extreme cases such as when $i = 7$, the number of terms in the numerator using Bareiss/Edmonds/Lipton algorithm $\#N_7$ balloons to $12,802$, while the actual number of terms we got

$\#f_7 = 4$, validating the efficiency of the approach in handling sparse rational systems where traditional matrix methods become computationally prohibitive.

5.4 Demo

We now present a demo of our implementation of the algorithm in Maple 2025 for solving the example system 5.3 presented in section 5.2.

```
|\^/|     Maple 2025 (X86 64 LINUX)
._|\|   |/|_. Copyright (c) Maplesoft, a division of Waterloo Maple
   Inc. 2025
\  MAPLE  /  All rights reserved. Maple is a trademark of
<____ ____>  Waterloo Maple Inc.
|       Type ? for help.

> Sys, Vars, params, num_vars, num_eqn:= get_data(test_case):
  Sys: y1x1+y_1x2=1,          y1y2x1-x2=1
> counter := 0:
> p:=107:
"B = ", proc(point_::list(integer), p::prime)
local A, T, subs_values, num_eqn, soln, L_eval;
global counter;
    counter := counter + 1;
    subs_values := zip((par, pnt) -> par = pnt, [y1, y2], point_);
    L_eval := Eval(L, subs_values) mod p;
    num_eqn := numelems({x1, x2});
    A := Matrix(num_eqn, num_eqn + 1, datatype = integer[8],
       L_eval);
    T := traperror(LinearAlgebra:-Modular:-LinearSolve(p, A, 1));
    if T = "matrix is singular" then return FAIL end if;
    soln := convert(A[1 .. num_eqn, num_eqn + 1], list);
    return soln
end proc
"MRFI  ========================================="
"MRFI Starting MRFI"
"MRFI Number of parameters:", 2
"MRFI Number of equations:", 2
"MRFI ========================================="
"MRFI numerator_done: ", [false, false]
"MRFI denominator_done: ", [false, false]
           "In NDSA"
           "T:= ", 4
"NDSA: alpha: ", [2, 3, 4, 5]
"NDSA:MQRFR failed. Trying again with more points"
```

```
"NDSA: mqrfr_status: ", [false, false]
"------------------------------------------------"
          "T:= ", 8
"NDSA: alpha: ", [2, 3, 4, 5, 6, 7, 8, 9]
          "In MQRFR"
"NDSA: MQRFR successful for component ", 1
"NDSA: MQRFR successful for component", 2
"NDSA: Termination condition met"
"MRFI num_points_mqrfr: ", [6, 5]
"MRFI max_num_points_mqrfr: ", 6
"in main evaluation loop of MRFI"
"MRFI T_old=", 1
"MRFI T=", 4
"MRFI sigma_[", 1, "] = ", [2, 3]

-------------------------------------------------
"Deterministic_NDSA: alpha: ", [2, 3, 4, 5, 6, 7]
"Deterministic_NDSA:Psi_alpha: ",[[2, 3], [3, 10], [4, 17], [5,
   24], [6, 31], [7, 38]]
"Deterministic_NDSA: Y: ",[[69, 92], [15, 21], [95, 39], [49, 101],
   [31, 94], [13, 33]]
"-------------------------------------------------
"MRFI sigma_[", 2, "] = ", [4, 9]

-------------------------------------------------
"Deterministic_NDSA: alpha: ", [2, 3, 4, 5, 6, 7]
"Deterministic_NDSA:Psi_alpha: ",[[2, 102], [3, 2], [4, 9], [5,
   16], [6, 23], [7, 30]]
"-------------------------------------------------
"MRFI sigma_[", 3, "] = ", [8, 27]

-------------------------------------------------
"Deterministic_NDSA: alpha: ", [2, 3, 4, 5, 6, 7]
"Deterministic_NDSA:Psi_alpha: ",  [[2, 92], [3, 99], [4, 106], [5,
   6], [6, 13], [7, 20]]
"Deterministic_NDSA: Y: ",[[35, 19], [17, 19], [62, 72], [67, 83],
   [49, 76], [36, 10]]
"-------------------------------------------------------""
MRFI sigma_[", 4, "] = ", [16, 81]

-------------------------------------------------
"Deterministic_NDSA: alpha: ", [2, 3, 4, 5, 6, 7]
"Deterministic_NDSA:Psi_alpha: ",[[2, 90], [3, 97], [4, 104], [5,
   4], [6, 11], [7, 18]]
"Deterministic_NDSA: Y: ",[[34, 20], [43, 100], [85, 49], [104,
   46], [45, 80], [104, 49]]
"-------------------------------------------------------"
"MRFI sigma_[", 5, "] = ", [32, 29]
```

```
72 ------------------------------------------------
73 "Deterministic_NDSA: alpha: ", [2, 3, 4, 5, 6, 7]
74 "Deterministic_NDSA:Psi_alpha: ", [[2, 33], [3, 40], [4, 47], [5,
      54], [6, 61], [7, 68]]
75 "Deterministic_NDSA: Y: ",[[12, 42], [102, 41], [16, 11], [89, 61],
      [26, 99], [49, 104]]
76 "------------------------------------------------------------"
77 "MRFI sigma_[", 6, "] = ", [64, 87]
78 ------------------------------------------------
79     "In Deterministic_NDSA"
80 "Deterministic_NDSA: alpha: ", [2, 3, 4, 5, 6, 7]
81 "Deterministic_NDSA:Psi_alpha: ",[[2, 81], [3, 88], [4, 95], [5,
      102], [6, 2], [7, 9]]
82 "Deterministic_NDSA: Y: ",[[22, 32], [28, 8], [29, 105], [16, 27],
      [92, 33], [86, 67]]
83 "------------------------------------------------------------"
84 "MRFI sigma_[", 7, "] = ", [21, 47]
85 ------------------------------------------------
86 "Deterministic_NDSA: alpha: ", [2, 3, 4, 5, 6, 7]
87 "Deterministic_NDSA:Psi_alpha: ",[[2, 21], [3, 28], [4, 35], [5,
      42], [6, 49], [7, 56]]
88 "Deterministic_NDSA: Y: ",[[61, 100], [81, 62], [26, 1], [21, 22],
      [61, 64], [17, 29]]
89 "------------------------------------------------------------"
90 "MRFI num_eval: ",[[92, 31, 16, 93, 33, 20, 101, 49], [107, 92, 47,
      19, 42, 4, 104, 83]]
91 "MRFI den_eval: ", [[92, 2, 67, 34, 45, 28, 61, 75], [92, 1, 97,
      31, 63, 41, 16, 80]]
92 "------------------------------------------------------------
93 "numerator_done: ", [false, false]
94 "denominator_done: ", [false, false]
95 "------------------------------------------------------"
96 "MRFI numerator_done[", 1, "]=", false
97 "------------------------------------------------
98           "In BMEA"
99 "Checking termination condition for numerator"
100 "MRFI Numerator component ", 1, " recovered!"
101 "------------------------------------------------
102 "MRFI numerator_done[", 2, "]=", false
103           "In BMEA"
104 "Checking termination condition for numerator"
105 "MRFI Numerator component ", 2, " recovered!"
106 "MFRI processing denominators now"
107 "MRFI common_den_flag: ", false
```

```
"Checking termination condition for denominator"
"MRFI Denominator component ", 1, " recovered!"
"-----------------------------------------------
          "In BMEA"
"Checking termination condition for denominator"
"MRFI Denominator component ", 2, " recovered!"
"-----------------------------------------------
"BMEA done status: ", [true, true]
   "All done status: ", true
"MRFI All components recovered!"
"MRFI Roots_num: ", [[1, 2], [1, 3]]
"MRFI Roots_den: ", [[2, 12], [1, 6]]
"-----------------------------------"
"In generate_monomials"
"In Zippel_Transpose_Vandermonde_solver"
"In construct_final_polynomial"
"MRFI ======================================="
"MRFI RECOVERY COMPLETE"
"MRFI ======================================="
     "numerator = ", 1+y1
 "denominator = ", y1^2*y2+y1
    "numerator = ", -1+y2
  "denominator = ", y1*y2+1
"======================================================="
    "Displaying the results"
          "x", 1, "="
"Rat_recon= ", (1+y1)/(y1^2*y2+y1)
"original_soln =", (1+y1)/y1/(y1*y2+1)
diff = 0
          "x", 2, "="
"Rat_recon= ", (-1+y2)/(y1*y2+1)
"original_soln =", (-1+y2)/(y1*y2+1)
diff = 0
"======================================================="
> print("Total number of lines generated in
   get_point_on_affine_line:", num_lines):
"Total number of lines generated in get_point_on_affine_line:", 9
> lprint("Total Black Box Calls:", counter):
"Total Black Box Calls:", 54
> quit
memory used=9.1MB, alloc=41.3MB, time=0.12
```

Bibliography

[1] Erwin H. Bareiss. Sylvester's identity and multistep integer-preserving gaussian elimination. *Mathematics of Computation*, 22(103):565–578, 1968.

[2] M. Ben-Or and P. Tiwari. A deterministic algorithm for sparse multivariate polynomial interpolation. In *Proceedings of STOC '20*, pages 301–309. ACM Press, 1988.

[3] Richard E. Blahut. *Theory and Practice of Error Control Codes.* Addison-Wesley, Reading, MA, 1983.

[4] Daniel Boley. Vandermonde factorization of a hankel matrix? *Linear Algebra and its Applications*, 263:11–23, 1997.

[5] David G. Cantor and Hans Zassenhaus. A new algorithm for factoring polynomials over finite fields. *Mathematics of Computation*, 36(154):587–592, 1981.

[6] David Cox, John Little, and Donal O'Shea. *Ideals, Varieties, and Algorithms.* Springer, 4 edition, 2015.

[7] D.S. Dummit and R.M. Foote. *Abstract Algebra.* Wiley, 2003.

[8] Jack Edmonds. Systems of distinct representatives and linear algebra. In *Proceedings of the American Mathematical Society*, volume 18, pages 787–793. American Mathematical Society, 1967.

[9] Shuai Geng, Jie Zhang, and Yuanming Zhang. On hankel operators on mixed norm spaces and a related operator equation. *arXiv preprint arXiv:2206.04126*, 2022.

[10] Robin Hartshorne. *Algebraic Geometry*, volume 52 of *Graduate Texts in Mathematics.* Springer-Verlag, 1977.

[11] Ayoola Jinadu and Michael B. Monagan. Solving parametric linear systems using sparse rational function interpolation. In François Boulier, Matthew England, Ilias S. Kotsireas, Timur M. Sadykov, and Evgenii V. Vorozhtsov, editors, *Computer Algebra in Scientific Computing - 25th International Workshop, CASC 2023, Havana, Cuba, August 28 - September 1, 2023, Proceedings*, volume 14139 of *Lecture Notes in Computer Science*, pages 233–254. Springer, 2023.

[12] Erich Kaltofen and Wen-shin Lee. Early termination in sparse interpolation algorithms. *Journal of Symbolic Computation*, 36(3–4):365–400, 2003.

[13] Erich L. Kaltofen and Barry M. Trager. Computing with polynomials given by black boxes for their evaluations: Greatest common divisors, factorization, separation of numerators and denominators. *Journal of Symbolic Computation*, 9(3):301–320, 1990.

[14] Erich L. Kaltofen and Zhengfeng Yang. On exact and approximate interpolation of sparse rational functions. In *Proceedings of the 2007 International Symposium on Symbolic and Algebraic Computation (ISSAC 07)*, pages 203–210, Waterloo, Ontario, Canada, 2007. ACM.

[15] John D. Lipson. Symbolic methods for the computer solution of linear equations with applications to flowgraphs. In R. Tobey, editor, *Proceedings of the 1968 Summer Institute on Symbolic Mathematical Computation*, pages 233–303, 1969.

[16] J. L. Massey. Shift-register synthesis and bch decoding. *IEEE Transactions on Information Theory*, 15:122–127, 1969.

[17] Michael Monagan. Maximal quotient rational reconstruction: An almost optimal algorithm for rational reconstruction. In *Proceedings of the 2004 International Symposium on Symbolic and Algebraic Computation (ISSAC 2004)*, pages 243–249, New York, NY, USA, 2004. ACM.

[18] Les Piegl and Wayne Tiller. *The NURBS Book*. Monographs in Visual Communication. Springer-Verlag, Berlin, Heidelberg, 2nd edition, 1997.

[19] Joachim von zur Gathen and Jürgen Gerhard. *Modern Computer Algebra*. Cambridge University Press, Cambridge, 3 edition, 2013.

[20] P. S. Wang, M. J. T. Guy, and J. H. Davenport. P-adic reconstruction of rational numbers. *SIGSAM Bulletin*, 16(2):2–3, 1982.

[21] Richard Zippel. Interpolating polynomials from their values. *Journal of Symbolic Computation*, 9:375–403, 1990.

Appendix A

Code

Appendices should be used for supplemental information that does not form part of the main research. Remember that figures and tables in appendices should not be listed in the List of Figures or List of Tables.

```
##########################################################
# 1.MFRI
##########################################################
 MRFI:=proc(B, num_vars::integer, num_eqn::integer, vars::list,
    p::integer, coeff_recovery::boolean, print_flag::boolean)
    local i, j, k, Primes, direction, sigma_, num_eval, den_eval,
       u, mon,
          numerator_done, denominator_done, T, T_old,
          mqrfr_results, lin_sys, num_points_mqrfr,
          Numerators, Denominators, deg_num, deg_den,
          lambda_num, lambda_den, terms_num, terms_den,
          R_num, R_den, Roots_num_eval, Roots_den_eval,
          num_mono, den_mono, coeff_num, coeff_den,
          final_num, final_den, r_, temp, common_den_flag,
          den_lc, den_lc_inv,bmea_done,r,temp_den,
          all_den_done,all_done,max_num_points_mqrfr:

    r_ := rand(p):
    Primes := [seq(ithprime(i), i=1..num_vars)]:
     direction := [seq(r_(), i=1..num_vars-1)]:
    sigma_ := []:

    # Initializing accumulators and flags
    num_eval := [seq([], i=1..num_eqn)]:
    den_eval := [seq([], i=1..num_eqn)]:

    numerator_done := [seq(false, i=1..num_eqn)]:
    denominator_done := [seq(false, i=1..num_eqn)]:
    bmea_done := [seq(false, i=1..num_eqn)]:
    all_done := true:

    # Initialize hash tables for lambda, terms, and R evaluations
    lambda_num := table():
```

```
    terms_num := table():
    R_num := table():
    lambda_den := table():
    terms_den := table():
    R_den := table():
    final_num := table():
    final_den := table():
    # Initialize hash tables for monomials, coefficients, and final
       polynomials
    num_mono := table():
    coeff_num := table():
    den_mono := table():
    coeff_den := table():

    # Initialize all entries from 1 to num_eqn
    for i from 1 to num_eqn do
        lambda_num[i] := []:
        terms_num[i] := []:
        R_num[i] := []:
        lambda_den[i] := []:
        terms_den[i] := []:
        R_den[i] := []:
    end do:

    T := 4:
    T_old := 1:
    num_points_mqrfr:=[seq(0, i=1..num_eqn)]:

    # Initial NDSA call
    mqrfr_results, lin_sys := NDSA(B, [seq(1, i=1..num_vars)],
       direction, num_vars, p, T,num_eqn,false):


    # Process initial results
    Numerators := [seq(mqrfr_results[i][1],
       i=1..nops(mqrfr_results))]:
    Denominiators := [seq(mqrfr_results[i][2],
       i=1..nops(mqrfr_results))]:

    for k from 1 to num_eqn do
        num_eval[k] := [eval(Numerators[k], x=1)]:
        den_eval[k] := [eval(Denominiators[k], x=1)]:
    end do:

    deg_num := [seq(degree(Numerators[i], x),
       i=1..nops(Numerators))]:
    deg_den := [seq(degree(Denominiators[i], x),
       i=1..nops(Denominiators))]:

    for i from 1 to numelems(deg_den) do
        num_points_mqrfr[i] := deg_num[i] + deg_den[i] + 2:
        # num_points_mqrfr[i] := deg_num[i] + deg_den[i] + 1:
    end do:
```

```
    max_num_points_mqrfr := max(op(deg_num)) + max(op(deg_den)) + 2:
    # max_num_points_mqrfr := max(op(deg_num)) + max(op(deg_den)) +
        1:

    # Check for a common denominator
    common_den_flag := true:

    for k from 2 to num_eqn do
        if Denominiators[k] <> Denominiators[1] then
            common_den_flag := false:
            break:
        end if:
    end do:

    # Main evaluation loop
    while true do
        for j from T_old to 2*T-1 do
            sigma_ := [op(sigma_), [seq(Primes[i]^j mod p,
                i=1..nops(Primes))]]:
            if print_flag then print("MRFI sigma_[",j,"] =
                ",sigma_[j]): end if;
            if lin_sys then
                mqrfr_results:= Deterministic_NDSA(B, sigma_[j],
                    direction, num_vars, p,
                    num_points_mqrfr,max_num_points_mqrfr,num_eqn,
                mqrfr_results,lin_sys:=NDSA(B,sigma_[j],direction,
                num_vars,p,max_num_points_mqrfr,
                max_num_points_mqrfr,1,false):
            end if:
            Numerators := [seq(mqrfr_results[i][1],
                i=1..nops(mqrfr_results))]:
            Denominators := [seq(mqrfr_results[i][2],
                i=1..nops(mqrfr_results))]:
            for k from 1 to num_eqn do
                if not numerator_done[k] then
                    num_eval[k] := [op(num_eval[k]),
                        eval(Numerators[k], x=sigma_[j][1]) mod p]:
                end if:
                if not denominator_done[k] then
                    den_eval[k] := [op(den_eval[k]),
                        eval(Denominiators[k], x=sigma_[j][1]) mod
                        p]:
                end if:
            end do:
        end do:

        # Apply BMEA
        all_done := true:
        for k from 1 to num_eqn do
```

```
            if  numerator_done[k] then if print_flag then
                print("MRFI Skipping BMEA for numerator component
                ",k," as already done"): end if; next: end if;
                # temp := BMEA(num_eval[k], p, Z):
                # lprint("MRFI temp (numerator)=",temp):

                # Use hash table
                lambda_num[k] := BMEA(num_eval[k], p, Z):
                terms_num[k] := degree(lambda_num[k], Z):
                R_num[k] := Roots(lambda_num[k]) mod p:

                # Add check for empty R_num[k]
                if R_num[k] = [] then
                    numerator_done[k] := false:
                    next:
                end if:
                if nops(R_num[k]) > 0 and R_num[k][1][1] = 0 then
                    R_num[k] := remove(x->x=[0,1], R_num[k]):
                    terms_num[k] := terms_num[k] - 1:
                end if:
                if nops(R_num[k]) = terms_num[k] and terms_num[k]
                   <= T then
                    numerator_done[k] := true:
                end if:
        end do:
        # Process denominators
        all_den_done := true:
        if common_den_flag then
            if not denominator_done[1] then
                # temp := BMEA(den_eval[1], p, Z):
                lambda_den[1] := BMEA(den_eval[1], p, Z):
                terms_den[1] :=degree(lambda_den[1], Z):
                R_den[1] := Roots(lambda_den[1]) mod p:
                if nops(R_den[1]) > 0 and R_den[1][1][1] = 0 then
                    R_den[1] := remove(x->x=[0,1], R_den[1]):
                    terms_den[1] := terms_den[1] - 1:
                end if:

                if nops(R_den[1]) = terms_den[1] and terms_den[1] <
                   T then
                    for k from 1 to num_eqn do
                        denominator_done[k] := true:
                    end do:
                end if:
            end if:
        else
            for k from 1 to num_eqn do
                if  denominator_done[k] then
                    next:
                end if;
                # temp := BMEA(den_eval[k], p, Z):

                # Use hash table
```

```
                lambda_den[k] := BMEA(den_eval[k], p, Z):
                terms_den[k] := degree(lambda_den[k], Z):
                R_den[k] := Roots(lambda_den[k]) mod p:

                    if nops(R_den[k]) > 0 and R_den[k][1][1] = 0
                       then
                        R_den[k] := remove(x->x=[0,1], R_den[k]):
                        terms_den[k] := terms_den[k] - 1:
                    end if:

                    if nops(R_den[k]) = terms_den[k] and
                       terms_den[k] < T then
                        denominator_done[k] := true:
                    end if:
            end do:
        end if:
        for i from 1 to num_eqn do
            bmea_done[i] := numerator_done[i] and
               denominator_done[i]:
        end do:
        for i from 1 to num_eqn do
            all_done:=all_done and bmea_done[i]:
        end do:
        if all_done then
            break:
        end if:
        T_old := 2*T:
        T := T*2:
        if( T > 2^6) then break; end if; # Safety break to avoid
           infinite loops during testing
    end do:
    # Extract roots and build polynomials
    Roots_num_eval := [seq([seq(r[1], r in R_num[k])],
       k=1..num_eqn)]:
    if common_den_flag then
        Roots_den_eval := [[seq(r[1], r in R_den[1])]]:
        for k from 2 to num_eqn do
            Roots_den_eval := [op(Roots_den_eval),
               Roots_den_eval[1]]:
        end do:
    else
        Roots_den_eval := [seq([seq(r[1], r in R_den[k])],
           k=1..num_eqn)]:
    end if:
    # Generate monomials and coefficients using hash tables
       (already initialized at the start)
    for k from 1 to num_eqn do
        if coeff_recovery = false  then
            temp := generate_monomials(Roots_num_eval[k], num_vars,
               Primes, vars):

            if temp = FAIL then return FAIL: end if:
            # Store in hash table with key k
```

```
213             num_mono[k] := temp:
214         end if:
215         coeff_num[k] :=
               Zippel_Transpose_Vandermonde_solver(num_eval[k],
               terms_num[k],
216                                       Roots_num_eval[k],
                                             lambda_num[k], p):
217     end do:
218     # Generate denominators using hash tables (already initialized
           at the start)
219     if common_den_flag then
220         if coeff_recovery = false  then
221             temp := generate_monomials(Roots_den_eval[1], num_vars,
                   Primes, vars):
222             if temp = FAIL then return FAIL: end if:
223             # Store common monomial list for first denominator
224             den_mono[1] := temp:
225         end if:
226         coeff_den[1] :=
               Zippel_Transpose_Vandermonde_solver(den_eval[1],
               terms_den[1],
227                                                 Roots_den_eval[1],
                                                       lambda_den[1], p):
228         temp_den := construct_final_polynomial(coeff_den[1],
               den_mono[1]):
229         # Set the same denominator for all equations
230         for k from 1 to num_eqn do
231             final_den[k] := temp_den:
232             if k > 1 then
233                 den_mono[k] := den_mono[1]:
234                 coeff_den[k] := coeff_den[1]:
235             end if:
236         end do:
237     else
238         for k from 1 to num_eqn do
239             if coeff_recovery = false  then
240                 temp := generate_monomials(Roots_den_eval[k],
                       num_vars, Primes, vars):
241                 if temp = FAIL then return FAIL: end if:
242                 # Store in hash table with key k
243                 den_mono[k] := temp:
244             end if:
245
246             coeff_den[k] :=
                   Zippel_Transpose_Vandermonde_solver(den_eval[k],
                   terms_den[k],
247                                                 Roots_den_eval[k],
                                                       lambda_den[k], p):
248         end do:
249     end if:
250     for k from 1 to num_eqn do
251         u:=(1/coeff_den[k][-1]) mod p:
252         coeff_num[k]:=u*coeff_num[k] mod p:
```

```
        coeff_den[k]:=u*coeff_den[k] mod p:
    end do:
    if coeff_recovery = false then
        return coeff_num,
           num_mono,coeff_den,den_mono,num_points_mqrfr,
        max_num_points_mqrfr,terms_num,terms_den:
    else
        return coeff_num, coeff_den:
    end if;
end proc:
```

```
  ##########################################################
# 2. NDSA - For Early stopping
##########################################################

with(LinearAlgebra):

NDSA:=proc(B,sigma_,beta_,num_var,p,num_points,num_eqn)
    local correct_degree,T,alpha,m,phi_,Psi_alpha,Y,u,f,g,dq,lcg,np,
    nv,i,r,
          lin_sys,temp,result,count,M,row,col,DQ,MQRFR_done:
    print("In NDSA");
    # print("NDSA: num_eqn:= ",num_eqn);
    MQRFR_done:=[seq(false, i=1..num_eqn)]:
    correct_degree:=false:
    lin_sys:=false:
    T:=num_points:
    temp:=[]:
    # result:=[]:
    result:=[seq([], i=1..num_eqn)]:
    count:=0:
    while(not(correct_degree)) do
        count:=count+1:

        print("T:= ",T):
        r:=rand(p):
        # r:=rand(2^31-1):
        # alpha:=[seq(r() mod p,i=1..T)]:
        alpha:=[seq(i mod p,i=1..T)]:# for example in Ch4: test
           case 1
        # alpha:=[seq(i ,i=2..T+1)]:
        lprint("NDSA: alpha: ",alpha):
        m:=expand(product(x-alpha[j],j=1..T)) mod p:
        lprint("NDSA:m: ",m):
        Psi_alpha:=get_point_on_affine_line(num_var,alpha,beta_,
        sigma_,p,T):
        lprint("NDSA:Psi_alpha: ",Psi_alpha):
        Y:=[seq(B(Psi_alpha[i],p),i=1..T)]:
        print("NDSA: Y = ",Y);
        # M:=convert(Y,Matrix):
        M:=Matrix(Y):
        # lprint("Matrix M: ",M):
        row,col:=Dimension(M):
```

```
        lprint("row: ",row, " col: ",col):

        if row =1 then
            lin_sys:=false:
            u:=Interp(alpha,Y,x)mod p:
            # lprint("NDSA: Single equation case - u: ",u):
            result:=[[MQRFR(m,u,0,1,p)]]:
            # lprint("result =",result):
            dq:=result[1][3]:
            print("NDSA: dq: ",dq):
        else
            lin_sys:=true:
            u:=get_u(M,col,alpha,p):
        end if:
        # lprint("NDSA: u: ",u):
        # print("nops u: ",nops(u));

        if lin_sys = true then
            for i from 1 to nops(u) do
            # put a check for MQRFR_done
                if(MQRFR_done[i]) then
                    print("NDSA: Skipping MQRFR for equation ",i,"
                        as already done"):
                    next:
                end if;
                temp:=[op(temp),MQRFR(m,u[i],0,1,p)]:
                # print("NDSA: result temp before assignment:
                    ",result):
                # print("NDSA: temp",i,": ",temp):
                # result:=[op(result),temp]:
                result[i]:=temp:
                print("NDSA: result",i,": ",result[i]):
                temp:=[]:
            end do:
            # i:=1:
            # print("i=",i):
            # lprint("NDSA:entire result: ",result):
            # result_list := convert(result,list):

            DQ:=[seq(result[i][3],i=1..nops(result))]:
            for i from 1 to numelems(DQ) do
                if DQ[i] > 1 then
                    MQRFR_done[i]:=true:
                    print("NDSA: MQRFR successful for equation ",i):
                end if;
            end do:
            # lprint("NDSA:DQ: ",DQ):
            dq:=min(DQ):
            # lprint("NDSA:Minimum dq: ",dq):
        end if:

        if dq > 1 then
            print("NDSA: Termination condition met"):
```

```
            return result,lin_sys:
            # if num_points <> T then
            #     return result,T,lin_sys:
            # else
            #     return result,lin_sys:
            # end if:
        else
            print("NDSA:MQRFR failed. Trying again with more
                points"):
            T:=T*2:
            # result:=[]:
            print("NDSA: mqrfr_status: ",MQRFR_done):
            for i from 1 to num_eqn do
                if MQRFR_done[i]= false then
                    lprint("NDSA: Resetting result for equation
                        ",i):
                    result[i]:=[]:
                end if;
            end do;
            print("_____________________________________________"):
            # result:=[seq([], i=1..num_eqn)]:
            DQ:=[]:
        end if:
        # if(count= 4) then break: end if:
    if(T>2^5)then break; end if; # Safety break to avoid infinite
        loops during testing
    end do:
end proc:
```

```
#########################################################
# 2. NDSA - For Vector Black Boxes
#########################################################

with(LinearAlgebra):

Deterministic_NDSA := proc(B, sigma_, beta_, num_var, p,
    num_points, max_points, num_eqn)
    local correct_degree, T, alpha, m, phi_, _phi, Y, u, f, g,
          dq, lcg, np, nv, i, r, lin_sys, temp, result, count,
          M, row, col, DQ, MQRFR_done:

    lin_sys := false:
    T := max_points:
    temp := []:
    result := [seq([], i = 1..num_eqn)]:
    count := 0:

    r := rand(p):
    alpha := [seq(r() + r() mod p, i = 1..T)]:

    _phi := projection_image_phi(num_var, alpha, beta_, sigma_, p,
        T):
    Y := [seq(B(_phi[i], p), i = 1..T)]:
```

```
23
24      for i from 1 to num_eqn do
25
26          m := expand(product(x - alpha[j], j = 1..num_points[i]))
                mod p:
27
28          M := convert(Y[1..num_points[i]], Matrix):
29          row, col := Dimension(M):
30
31          if row = 1 then
32              u := Interp(alpha[1..num_points[i]],
33                          Y[1..num_points[i]],
34                          x) mod p:
35          else
36              lin_sys := true:
37              u := Deterministic_get_u(M, i,
38                                       alpha[1..num_points[i]],
39                                       p):
40          end if:
41
42          if lin_sys = false then
43              result := [MQRFR(m, u, 0, 1, p)]:
44              dq := result[1][3]:
45          else
46              result[i] := [MQRFR(m, u, 0, 1, p)]:
47          end if;
48
49      end do:
50
51      return result:
52
53  end proc:
```

```
1
2
3   ############################################################
4   # 3. GET POINT ON AFFINE LINE PSI
5   ############################################################
6
7   get_point_on_affine_line := proc(num_var::posint, alpha::list,
        beta_::list,sigma_::list, p::prime, T::posint)
8       description "Generates evaluation points on a parametric affine
            line in finite field Z_p^n":
9
10      option remember;  # Cache results for repeated calls with same
            parameters
11
12  # 1. Parameters:
13  # -----------
14  # num_var : positive integer
15  #     The dimension of the affine space (number of variables in the
        polynomial system)
16  #     Constraint: num_var >= 2
```

```
#
# alpha : list of integers. These are points in Z_p. Map Psi will find the points on the line for these alphas.
#     List of T parameter values for the affine line, typically [alpha_1, alpha_2, ..., alpha_T]
#     These serve as the univariate parameter for the line parameterization
#     Requirement: nops(alpha) >= T, all elements in range [0, p-1]
#
# beta_ : list of integers
#     Direction vector coefficients [beta_1, beta_2, ..., beta_{num_var-1}]
#     Defines the direction of the affine line in the finite field space
#     Requirement: nops(beta_) = num_var - 1, all elements in range [0, p-1]
#
# sigma_ : list of integers
#     Base point coordinates [sigma_1, sigma_2, ..., sigma_{num_var}]
#     Defines the initial point through which the affine line passes
#     Requirement: nops(sigma_) = num_var, all elements in range [0, p-1]
#
# p : prime number
#     The characteristic of the finite field Z_p
#     All arithmetic operations are performed modulo p
#
# T : positive integer
#     Number of evaluation points to generate on the affine line
#     Constraint: T <= p (to ensure distinct points)
#
# 2. Returns:
# --------
# list of lists
#     A list of T points, where each point on the affine line is a list
#     Format: [[psi_1,1, psi_1,2, ..., psi_1,n],...,[psi_T,1, psi_T,2, ..., psi_T,n]]
#     where psi_i,j represents the j-th coordinate of the i-th point
#
# 3. Mathematical Description:
# ------------------------
# This procedure computes points on an affine line in Z_p^n parameterized as:
#     psi(alpha) = sigma + alpha * beta
# where:
#     - sigma = (sigma_1, sigma_2, ..., sigma_n) is the base point
#     - beta is the direction vector
#     - alpha is the line parameter
#
# Specifically, for each parameter value alpha_i:
```

```
58 #      psi_i,1 = alpha_i (first coordinate uses the parameter
       directly)
59 #      psi_i,j = beta_{j-1} * (alpha_i - sigma_1) + sigma_j (mod p)
       for j = 2, ..., num_var
60 #
61 # This creates a line that:
62 #      1. Passes through point sigma when alpha = sigma_1
63 #      2. Has direction determined by the beta coefficients
64 #      3. Ensures the first coordinate equals the parameter value
65 #
66 # 4. Algorithm Complexity:
67 # --------------------
68 # Time Complexity: O(T * num_var)
69 # Space Complexity: O(T * num_var)
70 #
71 # 5. Error Conditions:
72 # -----------------
73 # - If nops(alpha) < T: May cause index out of bounds error
74 # - If nops(beta_) != num_var - 1: Incorrect parameterization
75 # - If nops(sigma_) != num_var: Incorrect base point specification
76 #
77 # 6. Notes:
78 # ------
79 # The affine line construction ensures good separation of
       evaluation points
80 #
81 # 7. References:
82 # ----------
83
84     # [1] Kaltofen, E. & Yang, Z. (2007). "Sparse Multivariate
          Function Recovery"
85
86     local psi, nv, np, i;
87     global num_lines:
88     num_lines:=num_lines+1:
89     # Checking preconditions
90     if num_var < 2 then
91         error "Number of variables must be at least 2, got %1",
              num_var;
92     end if;
93     if nops(alpha) < T then
94         error "Insufficient alpha values: need %1, got %2", T,
              nops(alpha);
95     end if;
96     # Input validation (recommended to add)
97     ASSERT(num_var >= 2, "Number of variables must be at least 2");
98     ASSERT(nops(alpha) >= T, "Insufficient alpha values for T
          points");
99     ASSERT(nops(beta_) = num_var - 1, "beta_ must have exactly
          num_var-1 elements");
100    ASSERT(nops(sigma_) >= num_var, "sigma_ must have at least
          num_var elements");
101
```

```
102
103     # Generate T points on the affine line
104     for np from 1 to T do
105         # First coordinate is always the parameter value
106         psi[np][1] := alpha[np];
107
108         # Remaining coordinates follow the affine line formula
109         for nv from 2 to num_var do
110             psi[np][nv] := beta_[nv-1]*alpha[np] -
                    beta_[nv-1]*sigma_[1] + sigma_[nv] mod p;
111         end do;
112     end do;
113
114     # Convert the array of points to a list of lists
115     return [seq(convert(psi[i], list), i=1..T)];
116 end proc:
```

```
1  ##########################################################
2  # 5. BMEA - Berlekamp-Massey Extended Algorithm
3  ##########################################################
4
5  BMEA := proc(v::list,p::posint,Z::name)
6      description "Implements the Berlekamp-Massey Extended Algorithm
           to find the minimal polynomial of a given sequence":
7  # 1. Parameters:
8  # -----------
9  # v : list
10 #     A list of integers representing the sequence for which the
       minimal polynomial is to be found.
11 # p : positive integer
12 #     A prime number representing the modulus for arithmetic
       operations.
13 # Z : name
14 #     A variable name used in polynomial expressions.
15 # 2. Returns:
16 # --------
17 # polynom
18 #     The minimal polynomial of the input sequence v over the
       finite field Z_p.
19 # 3. Mathematical Description:
20 # ------------------------
21 # The Berlekamp-Massey Extended Algorithm is an efficient method
       for finding the minimal polynomial
22 # of a given sequence. The minimal polynomial is the polynomial of
       least degree that generates the sequence
23 # when evaluated at successive integers. The algorithm iteratively
       updates the candidate polynomial
24 # based on discrepancies observed in the sequence, ensuring that
       the polynomial remains minimal.
25 # 4. Algorithm Complexity:
26 # --------------------
27 # Time Complexity: O(n^2) where n is the length of the input
       sequence v.
```

```
# Space Complexity: O(n) for storing intermediate polynomials.
# 5. References:
# ----------
# [1] Berlekamp, E. R. (1968). "Algebraic Coding Theory."
# [2] Massey, J. L. (1969). "Shift-register synthesis and BCH
    decoding."


    local n,m,R0,R1,V0,V1,i,Q:

    print("In BMEA");
    lprint("v=",v);
    n := iquo( nops(v), 2 ):
    # lprint("n=",n);
    m := 2*n-1:
    # lprint("m=",m);
    R0 := Z^(2*n):
    # lprint("R0=",R0);
    R1 := add( v[m+1-i]*Z^i, i=0..m ) mod p:
    # lprint("R1=",R1);
    V0 := 0:
    V1 := 1:
    while n <= degree(R1,Z) do
        R0,R1 := R1,Rem(R0,R1,Z,'Q') mod p:
        # lprint("R0=",R0);
        # lprint("R1=",R1);

        V0,V1 := V1,Expand(V0-Q*V1) mod p:
        # lprint("V0=",V0);
        # lprint("V1=",V1);
    od:
    i := 1/lcoeff(V1,Z) mod p:
    return i*V1 mod p:
end:
```

```
############################################################
# 6. GENERATE MONOMIALS
############################################################

generate_monomials:=proc(roots_,num_var,prime_points,vars)
    local m,mm,i,j,counter,M_,rem:
    M_:=Vector(numelems(roots_),0):
    print("In generate_monomials"):
    print("roots_=",roots_):

    for i from 1 to numelems(roots_) do
        if(roots_[i]=0)then
            print("roots_[",i,"] = 0"):
            return FAIL:
        end if:
        mm:=roots_[i]:
        m:=1:
        for j from 1 to numelems(prime_points) do
```

```
            counter:=0:
            while mm mod prime_points[j] = 0 do
                mm:=iquo(mm,prime_points[j],'rem'):
                counter:=counter+1:
            end do:
            m:=m*vars[j]^counter:
        end do:
        M_[i]:=m:
    end do:

    if mm<> 1 then
        print("Warning: mm=",mm," (should be 1)"):
        return FAIL:
    end if:

    return convert(M_,list):
end proc:
```

```
s############################################################
# 7. Zippel Transpose Vandermonde Solver
############################################################

Zippel_Transpose_Vandermonde_solver:=proc(y::list,terms::integer,
roots_::list,lambda_::polynom,p::integer)
    description "Solves the Zippel Transpose Vandermonde system to
       find the coefficients of the polynomial":
# 1. Parameters:
# -----------
# y : list
#     A list of integers representing the evaluations of the
    univariate images of the numerator or denominator at  powers of
    2.
# terms : integer
#     The number of terms (degree + 1) in the polynomial to be
    reconstructed.
# roots_ : list
#     A list of integers representing the roots of the minimal
    polynomial obtained from the BMEA algorithm.
# lambda_ : polynom
#     The minimal characteristic polynomial obtained from the BMEA
    algorithm.
# p : integer
#     A prime number representing the modulus for arithmetic
    operations.
#
# 2. Returns:
# --------
# list
#     A list of coefficients of the polynomial reconstructed from
    the Vandermonde system in Z_p.
    local M,fin_coeff,q,q_lambda_inv,V_inv_b,i,j:
    lprint("In Zippel_Transpose_Vandermonde_solver"):
    M:=lambda_ mod p:
```

```
    fin_coeff:=Vector(terms,0):
    for i from 1 to terms do
        q:=quo(M,Z-roots_[i],Z):
        q_lambda_inv:= 1/ Eval(q,Z=roots_[i]) mod p:
        V_inv_b:=0:
        for j from 1 to terms do
            V_inv_b:=V_inv_b+coeff(q,Z,j-1)*y[j] mod p:
        end do:
        fin_coeff[i]:=V_inv_b*q_lambda_inv mod p:
    end do:
    # print("final_coeff: ",fin_coeff):
    return convert(fin_coeff,list):
end proc:
```

```
##########################################################
# 0. Main
##########################################################
read "./0_construct_BB.mpl":
read "./1_MRFI.mpl":
read "./2a_NDSA.mpl":
read "./2b_Deterministic_NDSA.mpl":
read "./3_get_point_on_affine_line.mpl":
read "./4_MQRFR.mpl":
read "./5_BMEA.mpl":
read "./6_generate_monomials.mpl":
read "./7_zippel_vandermonde_solver.mpl":
read "./8_construct_final_polynomial.mpl":
read "./data_gen.mpl":
read "./helpers.mpl":

# Vars,F,G,num_vars,num_eqn,params := get_data(1):
# counter := 0:
# num_lines:=0:
# B:= Construct_Rational_Blackbox(F,G,Vars):

# test_case:="rand":
# num_var:=3:
# num_terms:=11:
# den_terms:=9:
# Vars,F,G,num_vars,num_eqn,params:=get_data(test_case,num_var,
% num_terms,den_terms):
# counter := 0:
# num_lines:=0:
# B:= Construct_Rational_Blackbox(F,G,Vars):

# test_case:="rat_rand":
# num_var:=3:
# num_terms:=11:
# den_terms:=9:
# num_coeff_bound:=20:
# den_coeff_bound:=20:
# Vars,F,G,num_vars,num_eqn,params:=get_data(test_case,num_var,
#num_terms,den_terms,num_coeff_bound,den_coeff_bound):
```

```
# counter := 0:
# B:= Construct_Rational_Blackbox(F,G,Vars):

# lprint("Variables:", Vars):
# lprint("Numerator F:", F):
# lprint("Denominator G:", G):
# print("num_eqn =",num_eqn):

# test_case:="example":
# test_case:="small_sys_low_deg":
#  test_case:="bspline":
# test_case:="small_Sys":
test_case:="big_coeff":
# test_case:="mike":
# test_case:="bsbug":
num_lines:=0:

Sys, Vars, params, num_vars, num_eqn:= get_data(test_case):
print("Sys = ", Sys):
counter := 0:
B := Constuct_Sys_Blackbox(Sys, Vars, params):
print("B = ", B):

p:= 2^31 - 1:
# p:=107:
print("parameters are ", params):
print("Number of equations:", num_eqn):
print("Number of parameters:", num_vars):
# modular images modular_num[i][j]= fi mod pj where xi=fi/gi
modular_num:=table():
modular_den:=table():
for i from 1 to num_eqn do
    modular_num[i]:=table():
    modular_den[i]:=table():
end do:
Ratrecon_num:=table():
Ratrecon_den:=table():
Final_rat_poly:=table():

num_coeff_recovered:=[seq(false, i=1..num_eqn)]:
den_coeff_recovered:=[seq(false, i=1..num_eqn)]:
all_coeff_recovered:=true:
coeff_recovery:=false:
if coeff_recovery = false then
    try
    Num_coeff,Num_mono,Den_coeff,Den_mono,num_points_mqrfr,
    max_num_points_mqrfr,terms_num,terms_den := MRFI(B, num_vars,
       num_eqn,
    params, p,coeff_recovery,false):
        catch:
        print("ERROR:", lasterror()):
    end try:
```

```
91      print("Maximum number of points for MQRFR: ",
            max_num_points_mqrfr):
92      for i from 1 to num_eqn do
93          modular_num[i][1]:=construct_final_polynomial(Num_coeff[i],
                Num_mono[i]):
94          modular_den[i][1]:=construct_final_polynomial(Den_coeff[i],
                Den_mono[i]):
95          print("Modular numerator for equation ",i,": ",
                modular_num[i][1]):
96          print("Modular denominator for equation ",i,": ",
                modular_den[i][1]):
97          print("Number of points for MQRFR for equation ",i,": ",
                num_points_mqrfr[i]):
98          print("Terms in numerator for equation ",i,": ",
                terms_num[i]):
99          print("Terms in denominator for equation ",i,": ",
                terms_den[i]):
100     end do:
101     for i from 1 to num_eqn do
102         Ratrecon_num[i]:=iratrecon(modular_num[i][1],p):
103         if Ratrecon_num[i] <> FAIL then
                num_coeff_recovered[i]:=true:
104              print("numerator = ",Ratrecon_num[i]):
105         end if:
106         Ratrecon_den[i]:=iratrecon(modular_den[i][1],p):
107         if Ratrecon_den[i] <> FAIL then
                den_coeff_recovered[i]:=true:
108              print("denominator = ",Ratrecon_den[i]):
109         end if:
110         all_coeff_recovered := all_coeff_recovered and
                num_coeff_recovered[i] and den_coeff_recovered[i]:
111     end do:
112     if not all_coeff_recovered then coeff_recovery := true:  end if:
113 end if:
114
115 if coeff_recovery = true then
116     P:=[p]:
117     curr_p:=p:
118     M:=p:
119     end if:
120 while coeff_recovery = true do
121     print("Rational coefficient recovery failed "):
122     p_next:=nextprime(curr_p):
123     print("Trying next prime: ", p_next):
124     P:=[op(P), p_next]:
125     num_primes_used := numelems(P):
126     print("Number of primes used: ", num_primes_used):
127     M:=M*p_next:
128     for i from 1 to num_eqn do
129         try
130              Num_coeff,Den_coeff := MRFI(B, num_vars, num_eqn,
                    params, p_next,coeff_recovery,false):
131         catch:
```

```
132         print("ERROR:", lasterror()):
133     end try:
134         modular_num[i][num_primes_used]:=construct_final_polynomial
135         (Num_coeff[i], Num_mono[i]):
136         modular_den[i][num_primes_used]:=construct_final_polynomial
137         (Den_coeff[i], Den_mono[i]):
138         print("Modular numerator for equation ",i,": ",
               modular_num[i][num_primes_used]):
139         print("Modular denominator for equation ",i,": ",
               modular_den[i][num_primes_used]):
140     end do:
141     H_num:=[seq([seq(modular_num[i][j],j=1..num_primes_used)],
142     i=1..num_eqn)]:
143     H_den:=[seq([seq(modular_den[i][j],j=1..num_primes_used)],
144     i=1..num_eqn)]:
145     print("H_num so far: ", H_num):
146     print("H_den so far: ", H_den):
147     print("M so far: ", M):
148     print("Primes so far: ", P):
149     U_num:=[seq(chrem(H_num[i],P),i=1..num_eqn)]:
150     U_den:=[seq(chrem(H_den[i],P),i=1..num_eqn)]:
151     print("U_num so far: ", U_num):
152     print("U_den so far: ", U_den):
153     Ratrecon_num:=[seq(iratrecon(U_num[i],M),i=1..num_eqn)]:
154     Ratrecon_den:=[seq(iratrecon(U_den[i],M),i=1..num_eqn)]:
155     print("Numerators reconstructed so far: ", Ratrecon_num):
156     print("Denominators reconstructed so far: ", Ratrecon_den):
157     if not ormap(x -> x = FAIL, Ratrecon_num) and not ormap(x -> x
           = FAIL, Ratrecon_den) then
158         print("Rational coefficient recovery successful with primes
               ", P):
159         coeff_recovery := false:
160     else
161         print("Rational coefficient recovery failed with primes ",
               P):
162     end if:
163     curr_p:=p_next:
164     end do:
165 print("=======================================================");
166 print("Displaying the results");
167 if(num_eqn >1)then
168     og_soln:=get_eqn(Sys,Vars):
169     og_soln:=convert(og_soln,list):
170     for i from 1 to num_eqn do
171         print("x",i,"="):
172         Final_rat_poly[i]:=Ratrecon_num[i]/Ratrecon_den[i]:
173         lprint("Rat_recon= ",Final_rat_poly[i]):
174         lprint("original_soln =",op(2,og_soln[i])):
175         printf("f%d/g%d-ff%d/gg%d = %a\n",i,i,i,i,
176         simplify(Final_rat_poly[i]-op(2,og_soln[i])));
177     end do:
178     elif num_eqn =1 then
179         Final_rat_poly[1]:=Ratrecon_num[1]/Ratrecon_den[1]:
```

```
180         lprint("Rat_recon= ",Final_rat_poly[1]):
181         lprint("Original polynomial =",F/G):
182         printf("f1/g1 - F/G =
               %a\n",simplify(Final_rat_poly[1]-F/G));
183 end if:
184 print("=======================================================");
185 print("Total number of lines generated in
       get_point_on_affine_line:", num_lines):
186 lprint("Total Black Box Calls:", counter):
187
188 print("Total number of primes used for rational coefficient
       recovery: ", numelems(P)):
189 print("Primes used for rational coefficient recovery: ", P):
```