Modular algorithms for computing triangular decompositions of polynomial systems

Marc Moreno Maza

Ontario Research Center for Computer Algebra
Departments of Computer Science and Mathematics
University of Western Ontario, Canada

RTCA 2023, Institut Henri Poincaré, France, October 16

Acknowledgements

- Many thanks to the RTCA organizers for this event and for bringing all of us in this historical site.
- This talk is based on research projects in which many of my former and current graduate students have played an essential role. By alphabetic order: Alexander Brandt (Dalhousie University), Changbo Chen (CIGIT Chinese Academy of Sciences), Juan-Pablo Gonzàlez-Trochez (University of Western Ontario), François Lemaire (Université de Lille), Robert Moir (Earth64), Wei Pan (NVIDIA), Yuzhen Xie (Scotiabank), Haoze Yuan (University of Western Ontario).
- This talk is also based on collaborations with Maplesoft and the following colleagues: François Boulier (Université de Lille), Xavier Dahan (Tohoku University), Éric Schost (University of Waterloo), Wenyuan Wu (CIGIT Chinese Academy of Sciences).

Tentative Plan

- Part 1: Triangular decompositions in polynomial system solving
- Part 2: Modular methods in polynomial system solving
- Part 3: A modular method for triangular decompositions

Tentative Plan

- Part 1: Triangular decompositions in polynomial system solving
- Part 2: Modular methods in polynomial system solving
- Part 3: A modular method for triangular decompositions

Part 3 is based on

- our JSC 2012 paper with Changbo Chen ,and
- our recent CASC 2023 paper with Alexander Brandt, Juan-Pablo Gonzàlez-Trochez and Haoze Yuan.

Tentative Plan

- Part 1: Triangular decompositions in polynomial system solving
- Part 2: Modular methods in polynomial system solving
- Part 3: A modular method for triangular decompositions

Part 3 is based on

- our JSC 2012 paper with Changbo Chen ,and
- our recent CASC 2023 paper with Alexander Brandt, Juan-Pablo Gonzàlez-Trochez and Haoze Yuan.

A proof-of-concept implementation was done with the `RegularChains` library and an efficient implementation is under development in the BPAS library. See our CASC 2023 paper.

Tentative Plan

- Part 1: Triangular decompositions in polynomial system solving
- Part 2: Modular methods in polynomial system solving
- Part 3: A modular method for triangular decompositions

Part 3 is based on

- our JSC 2012 paper with Changbo Chen ,and
- our recent CASC 2023 paper with Alexander Brandt, Juan-Pablo Gonzàlez-Trochez and Haoze Yuan.

A proof-of-concept implementation was done with the `RegularChains` library and an efficient implementation is under development in the BPAS library. See our CASC 2023 paper.
These slides are available here.

What can the RegularChains library do for you?

- **Solving over $\mathbb{C}$**: that is, solving any system of multivariate polynomial equations (say, $f_1 = \cdots = f_m = 0$) and inequations $h \neq 0$:

What can the RegularChains library do for you?

- **Solving over $\mathbb{C}$**: that is, solving any system of multivariate polynomial equations (say, $f_1 = \cdots = f_m = 0$) and inequations $h \neq 0$:
  ↳ computing **all** its solutions symbolically, or only the **generic ones**

What can the RegularChains library do for you?

- **Solving over $\mathbb{C}$**: that is, solving any system of multivariate polynomial equations (say, $f_1 = \cdots = f_m = 0$) and inequations $h \neq 0$:
  - ↳ computing **all** its solutions symbolically, or only the **generic ones**
  - ↳ providing tools to **extract information** (dimension, degree, etc.) about those solutions and,

What can the `RegularChains` library do for you?

- **Solving over $\mathbb{C}$**: that is, solving any system of multivariate polynomial equations (say, $f_1 = \cdots = f_m = 0$) and inequations $h \neq 0$:
  - ↳ computing **all** its solutions symbolically, or only the **generic ones**
  - ↳ providing tools to **extract information** (dimension, degree, etc.) about those solutions and,
  - ↳ **performing** (set or geometric) **operations** on solutions sets.

What can the RegularChains library do for you?

- **Solving over $\mathbb{C}$**: that is, solving any system of multivariate polynomial equations (say, $f_1 = \cdots = f_m = 0$) and inequations $h \neq 0$:
  - ↳ computing **all** its solutions symbolically, or only the **generic ones**
  - ↳ providing tools to **extract information** (dimension, degree, etc.) about those solutions and,
  - ↳ **performing** (set or geometric) **operations** on solutions sets.
- **Solving over $\mathbb{R}$**: that is, solving any system of multivariate polynomial equations (say, $f_1 = \cdots = f_m = 0$, inequations $h \neq 0$ and inequalities $g_1 > 0, \ldots, g_p > 0$

What can the `RegularChains` library do for you?

- **Solving over $\mathbb{C}$**: that is, solving any system of multivariate polynomial equations (say, $f_1 = \cdots = f_m = 0$) and inequations $h \neq 0$:
  - ↳ computing **all** its solutions symbolically, or only the **generic ones**
  - ↳ providing tools to **extract information** (dimension, degree, etc.) about those solutions and,
  - ↳ **performing** (set or geometric) **operations** on solutions sets.
- **Solving over $\mathbb{R}$**: that is, solving any system of multivariate polynomial equations (say, $f_1 = \cdots = f_m = 0$, inequations $h \neq 0$ and inequalities $g_1 > 0, \ldots, g_p > 0$
  - ↳ doing the **same as above**, and

What can the RegularChains library do for you?

- **Solving over $\mathbb{C}$**: that is, solving any system of multivariate polynomial equations (say, $f_1 = \cdots = f_m = 0$) and inequations $h \neq 0$:
  - ↳ computing **all** its solutions symbolically, or only the **generic ones**
  - ↳ providing tools to **extract information** (dimension, degree, etc.) about those solutions and,
  - ↳ **performing** (set or geometric) **operations** on solutions sets.
- **Solving over $\mathbb{R}$**: that is, solving any system of multivariate polynomial equations (say, $f_1 = \cdots = f_m = 0$, inequations $h \neq 0$ and inequalities $g_1 > 0, \ldots, g_p > 0$
  - ↳ doing the **same as above**, and
  - ↳ finding **sample solutions**, and

What can the RegularChains library do for you?

- **Solving over $\mathbb{C}$**: that is, solving any system of multivariate polynomial equations (say, $f_1 = \cdots = f_m = 0$) and inequations $h \neq 0$:
  - ↳ computing **all** its solutions symbolically, or only the **generic ones**
  - ↳ providing tools to **extract information** (dimension, degree, etc.) about those solutions and,
  - ↳ **performing** (set or geometric) **operations** on solutions sets.

- **Solving over $\mathbb{R}$**: that is, solving any system of multivariate polynomial equations (say, $f_1 = \cdots = f_m = 0$, inequations $h \neq 0$ and inequalities $g_1 > 0, \ldots, g_p > 0$
  - ↳ doing the **same as above**, and
  - ↳ finding **sample solutions**, and
  - ↳ performing **cylindrical algebraic decomposition (CAD)** and **quantifier elimination (QE)**.

What can the RegularChains library do for you?

- **Solving over $\mathbb{C}$**: that is, solving any system of multivariate polynomial equations (say, $f_1 = \cdots = f_m = 0$) and inequations $h \neq 0$:
  - ↳ computing **all** its solutions symbolically, or only the **generic ones**
  - ↳ providing tools to **extract information** (dimension, degree, etc.) about those solutions and,
  - ↳ **performing** (set or geometric) **operations** on solutions sets.

- **Solving over $\mathbb{R}$**: that is, solving any system of multivariate polynomial equations (say, $f_1 = \cdots = f_m = 0$, inequations $h \neq 0$ and inequalities $g_1 > 0, \ldots, g_p > 0$
  - ↳ doing the **same as above**, and
  - ↳ finding **sample solutions**, and
  - ↳ performing **cylindrical algebraic decomposition (CAD)** and **quantifier elimination (QE)**.

- **Solving parametrically over $\mathbb{C}$ and over $\mathbb{R}$**: that is:

What can the RegularChains library do for you?

- **Solving over $\mathbb{C}$**: that is, solving any system of multivariate polynomial equations (say, $f_1 = \cdots = f_m = 0$) and inequations $h \neq 0$:
  - ↳ computing **all** its solutions symbolically, or only the **generic ones**
  - ↳ providing tools to **extract information** (dimension, degree, etc.) about those solutions and,
  - ↳ **performing** (set or geometric) **operations** on solutions sets.

- **Solving over $\mathbb{R}$**: that is, solving any system of multivariate polynomial equations (say, $f_1 = \cdots = f_m = 0$, inequations $h \neq 0$ and inequalities $g_1 > 0, \ldots, g_p > 0$
  - ↳ doing the **same as above**, and
  - ↳ finding **sample solutions**, and
  - ↳ performing **cylindrical algebraic decomposition (CAD)** and **quantifier elimination (QE)**.

- **Solving parametrically over $\mathbb{C}$ and over $\mathbb{R}$**: that is:
  - ↳ **finding conditions** on the parameters for the solutions to have a prescribed property (e.g. a unique real solution), or

What can the `RegularChains` library do for you?

- **Solving over $\mathbb{C}$**: that is, solving any system of multivariate polynomial equations (say, $f_1 = \cdots = f_m = 0$) and inequations $h \neq 0$:
  - ↳ computing **all** its solutions symbolically, or only the **generic ones**
  - ↳ providing tools to **extract information** (dimension, degree, etc.) about those solutions and,
  - ↳ **performing** (set or geometric) **operations** on solutions sets.

- **Solving over $\mathbb{R}$**: that is, solving any system of multivariate polynomial equations (say, $f_1 = \cdots = f_m = 0$, inequations $h \neq 0$ and inequalities $g_1 > 0, \ldots, g_p > 0$
  - ↳ doing the **same as above**, and
  - ↳ finding **sample solutions**, and
  - ↳ performing **cylindrical algebraic decomposition (CAD)** and **quantifier elimination (QE)**.

- **Solving parametrically over $\mathbb{C}$ and over $\mathbb{R}$**: that is:
  - ↳ **finding conditions** on the parameters for the solutions to have a prescribed property (e.g. a unique real solution), or
  - ↳ **computing** all or part of the **solutions** as functions of the parameters.

The design of the RegularChains library

- Hierarchy of the **user-interface**

The design of the RegularChains library

- Hierarchy of the **user-interface**
  - ↳ At the top-level, 29 commands for most common tasks, e.g. `PolynomialRing`, `Triangularize`, `RealTriangularize`, `Display`,

The design of the RegularChains library

- Hierarchy of the **user-interface**
  - ↳ At the top-level, 29 commands for most common tasks, e.g. PolynomialRing, Triangularize, RealTriangularize, Display,
  - ↳ 6 sub-packages for more specialized tasks: AlgebraicGeometryTools, ChainTools, ConstructibleSetTools, FastArithmeticTools, ParametricSystemTools, SemiAlgebraicSetTools.

The design of the RegularChains library

- Hierarchy of the **user-interface**
  - ↳ At the top-level, 29 commands for most common tasks, e.g. `PolynomialRing`, `Triangularize`, `RealTriangularize`, `Display`,
  - ↳ 6 sub-packages for more specialized tasks: `AlgebraicGeometryTools`, `ChainTools`, `ConstructibleSetTools`, `FastArithmeticTools`, `ParametricSystemTools`, `SemiAlgebraicSetTools`.
- Enforced but friendly **use of types**:

The design of the `RegularChains` library

- Hierarchy of the **user-interface**
  - ↳ At the top-level, 29 commands for most common tasks, e.g. `PolynomialRing`, `Triangularize`, `RealTriangularize`, `Display`,
  - ↳ 6 sub-packages for more specialized tasks: `AlgebraicGeometryTools`, `ChainTools`, `ConstructibleSetTools`, `FastArithmeticTools`, `ParametricSystemTools`, `SemiAlgebraicSetTools`.

- Enforced but friendly **use of types**:
  - ↳ Every `RegularChains` object has a type, e.g. `polynomial_ring`, `regular_chain`, `constructible_set`, `semi_algebraic_set`,

The design of the RegularChains library

- Hierarchy of the **user-interface**
  - ↳ At the top-level, 29 commands for most common tasks, e.g. `PolynomialRing`, `Triangularize`, `RealTriangularize`, `Display`,
  - ↳ 6 sub-packages for more specialized tasks: `AlgebraicGeometryTools`, `ChainTools`, `ConstructibleSetTools`, `FastArithmeticTools`, `ParametricSystemTools`, `SemiAlgebraicSetTools`.
- Enforced but friendly **use of types**:
  - ↳ Every `RegularChains` object has a type, e.g. `polynomial_ring`, `regular_chain`, `constructible_set`, `semi_algebraic_set`,
  - ↳ which ensures that the object of that type has properties,

The design of the RegularChains library

- Hierarchy of the **user-interface**
  - ↳ At the top-level, 29 commands for most common tasks, e.g. `PolynomialRing`, `Triangularize`, `RealTriangularize`, `Display`,
  - ↳ 6 sub-packages for more specialized tasks: `AlgebraicGeometryTools`, `ChainTools`, `ConstructibleSetTools`, `FastArithmeticTools`, `ParametricSystemTools`, `SemiAlgebraicSetTools`.
- Enforced but friendly **use of types**:
  - ↳ Every `RegularChains` object has a type, e.g. `polynomial_ring`, `regular_chain`, `constructible_set`, `semi_algebraic_set`,
  - ↳ which ensures that the object of that type has properties,
  - ↳ while the end-user does not need to explicitly manipulate this type.

The design of the RegularChains library

- Hierarchy of the **user-interface**
  - ↳ At the top-level, 29 commands for most common tasks, e.g. `PolynomialRing`, `Triangularize`, `RealTriangularize`, `Display`,
  - ↳ 6 sub-packages for more specialized tasks: `AlgebraicGeometryTools`, `ChainTools`, `ConstructibleSetTools`, `FastArithmeticTools`, `ParametricSystemTools`, `SemiAlgebraicSetTools`.
- Enforced but friendly **use of types**:
  - ↳ Every `RegularChains` object has a type, e.g. `polynomial_ring`, `regular_chain`, `constructible_set`, `semi_algebraic_set`,
  - ↳ which ensures that the object of that type has properties,
  - ↳ while the end-user does not need to explicitly manipulate this type.
- Criteria for selecting the **algorithms supporting the solvers**:

The design of the RegularChains library

- Hierarchy of the **user-interface**
  - ↳ At the top-level, 29 commands for most common tasks, e.g. `PolynomialRing`, `Triangularize`, `RealTriangularize`, `Display`,
  - ↳ 6 sub-packages for more specialized tasks: `AlgebraicGeometryTools`, `ChainTools`, `ConstructibleSetTools`, `FastArithmeticTools`, `ParametricSystemTools`, `SemiAlgebraicSetTools`.
- Enforced but friendly **use of types**:
  - ↳ Every `RegularChains` object has a type, e.g. `polynomial_ring`, `regular_chain`, `constructible_set`, `semi_algebraic_set`,
  - ↳ which ensures that the object of that type has properties,
  - ↳ while the end-user does not need to explicitly manipulate this type.
- Criteria for selecting the **algorithms supporting the solvers**:
  - ↳ provide a `comprehensive and coherent` set of tools for manipulating polynomial systems,

The design of the `RegularChains` library

- Hierarchy of the **user-interface**
  ↳ At the top-level, 29 commands for most common tasks, e.g. `PolynomialRing`, `Triangularize`, `RealTriangularize`, `Display`,
  ↳ 6 sub-packages for more specialized tasks: `AlgebraicGeometryTools`, `ChainTools`, `ConstructibleSetTools`, `FastArithmeticTools`, `ParametricSystemTools`, `SemiAlgebraicSetTools`.

- Enforced but friendly **use of types**:
  ↳ Every `RegularChains` object has a type, e.g. `polynomial_ring`, `regular_chain`, `constructible_set`, `semi_algebraic_set`,
  ↳ which ensures that the object of that type has properties,
  ↳ while the end-user does not need to explicitly manipulate this type.

- Criteria for selecting the **algorithms supporting the solvers**:
  ↳ provide a `comprehensive and coherent` set of tools for manipulating polynomial systems,
  ↳ implement solvers with both `general algorithms` (which may not be the most efficient ones) and `faster algorithms` (which may only work under some assumptions).

The BPAS library

Basic Polynomial Algebra Subprograms

http://www.bpaslib.org/

A high-performance polynomial algebra library

- Core of library written in C, wrapped in C++ interface for usability and object-oriented programming

Optimized algorithms and data structures, data locality, and parallelism

- Sparse multivariate polynomials [1], dense univariate and bivariate [7]
- Triangular decomposition of polynomial systems [2, 3]

User-friendly, object-oriented interface based on template meta-programming [6]

- A natural encoding of the algebraic hierarchy
- “Dynamic” creation of algebraic types through composition
- Compile-time type safety between algebraic types

Generic support for parallel programming and parallel patterns (this talk)

Outline

1. Triangular decompositions in polynomial system solving

2. Modular methods in polynomial system solving

3. A Modular methods for incremental triangular decompositions

4. Conclusions

Milestones (1/3)

- Let $\mathbf{k}$ be a field and $\mathbf{K}$ its algebraic closure. Consider n variables $x_1 < \cdots < x_n$.

Milestones (1/3)

- Let $\mathbf{k}$ be a field and $\mathbf{K}$ its algebraic closure. Consider n variables $x_1 < \cdots < x_n$.
- A subset $V \subset \mathbf{K}^n$ is a *(affine) variety over* $\mathbf{k}$ if there exists $F \subset \mathbf{k}[x_1, \ldots, x_n]$ such that $V = V(F)$ where

$$V(F) := \{z \in \mathbf{K}^n \ \mid \ f(z) = 0 \ \ (\forall \ f \in F)\}.$$

The variety V is *irreducible* if for all varieties $V_1, V_2 \subset \mathbf{K}^n$

$$V = V_1 \ \cup \ V_2 \quad \Rightarrow \quad V = V_1 \ \text{ or } \ V = V_2.$$

Milestones (1/3)

- Let $\mathbf{k}$ be a field and $\mathbf{K}$ its algebraic closure. Consider n variables $x_1 < \cdots < x_n$.
- A subset $V \subset \mathbf{K}^n$ is a *(affine) variety over* $\mathbf{k}$ if there exists $F \subset \mathbf{k}[x_1, \ldots, x_n]$ such that $V = V(F)$ where

$$V(F) := \{z \in \mathbf{K}^n \mid f(z) = 0 \quad (\forall\ f \in F)\}.$$

The variety V is *irreducible* if for all varieties $V_1, V_2 \subset \mathbf{K}^n$

$$V = V_1 \cup V_2 \quad \Rightarrow \quad V = V_1 \text{ or } V = V_2.$$

- **Theorem** (E. Lasker, 1905) *For each variety $V \subset \mathbf{K}^n$ there exist finitely many irreducible varieties $V_1, \ldots, V_e \subset \mathbf{K}^n$* such that

$$V = V_1 \cup \cdots \cup V_e.$$

Moreover, if $V_i \not\subseteq V_j$ for $1 \leq i < j \leq e$ then $\{V_1, \ldots, V_e\}$ is unique. This is the *irreducible decomposition of* V.

Milestones (2/3)

- **Theorem** (J.F. Ritt, 1932) *Let* $V \subset \mathbf{K}^n$ *be an* irreducible *non-empty variety and let* $F \subset \mathbf{k}[x_1, \ldots, x_n]$ *s.t.* $V = V(F)$. *Then, one can compute a (reduced) triangular set* $T \subset \langle F \rangle$ *s.t.*

$$(\forall\ g \in \langle F \rangle) \quad \mathrm{prem}(g, T) = 0.$$

  Combined with algebraic factorization one can (in theory) compute irreducible decompositions.

Milestones (2/3)

- **Theorem** (J.F. Ritt, 1932) *Let $V \subset \mathbf{K}^n$ be an* irreducible *non-empty variety and let $F \subset \mathbf{k}[x_1, \ldots, x_n]$ s.t. $V = V(F)$. Then, one can compute a (reduced) triangular set $T \subset \langle F \rangle$ s.t.*

$$(\forall\ g \in \langle F \rangle)\ \ \mathrm{prem}(g, T) = 0.$$

  Combined with algebraic factorization one can (in theory) compute irreducible decompositions.

- **Theorem** (W.T. Wu, 1987) *Let $V \subset \mathbf{K}^n$ be a variety and let $F \subset \mathbf{k}[x_1, \ldots, x_n]$ s.t. $V = V(F)$. Then, one can compute a (reduced) triangular set $T \subset \langle F \rangle$ s.t.*

$$(\forall\ g \in F)\ \ \mathrm{prem}(g, T) = 0.$$

  This leads to a factorization-free *algorithm for decomposing varieties (but not into irreducible components).*

Milestones (3/3)

- **Example.** Applying the `charset` procedure to $F = \{x_2^2 - x_1, x_1 x_3^2 - 2x_2x_3 + 1, (x_2x_3 - 1)x_4^2 + x_2^2\}$ produces $T = F$. However $V(F) = \varnothing$. Indeed

$$x_1x_3^2 - 2x_2x_3 + 1 \equiv (x_2x_3 - 1)^2 \mod x_2^2 - x_1.$$

Thus, the initial $(x_2x_3 - 1)$ is a zero-divisor modulo $\langle x_2^2 - x_1, x_1x_3^2 - 2x_2x_3 + 1\rangle$.

Milestones (3/3)

- **Example.** Applying the `charset` procedure to $F = \{x_2^2 - x_1, x_1x_3^2 - 2x_2x_3 + 1, (x_2x_3 - 1)x_4^2 + x_2^2\}$ produces $T = F$. However $V(F) = \varnothing$. Indeed

$$x_1x_3^2 - 2x_2x_3 + 1 \equiv (x_2x_3 - 1)^2 \mod x_2^2 - x_1.$$

  Thus, the initial $(x_2x_3 - 1)$ is a zero-divisor modulo $\langle x_2^2 - x_1, x_1x_3^2 - 2x_2x_3 + 1\rangle$.

- The notion of a *regular chain* (Lu Yang, Jingzhong Zhang 1991), (Michael Kalkbrener 1991), (Daniel Lazard 1991) solves this difficulty

Milestones (3/3)

- **Example.** Applying the `charset` procedure to $F = \{x_2^2 - x_1, x_1x_3^2 - 2x_2x_3 + 1, (x_2x_3 - 1)x_4^2 + x_2^2\}$ produces $T = F$. However $V(F) = \varnothing$. Indeed

$$x_1x_3^2 - 2x_2x_3 + 1 \equiv (x_2x_3 - 1)^2 \mod x_2^2 - x_1.$$

Thus, the initial $(x_2x_3 - 1)$ is a zero-divisor modulo $\langle x_2^2 - x_1, x_1x_3^2 - 2x_2x_3 + 1\rangle$.

- The notion of a *regular chain* (Lu Yang, Jingzhong Zhang 1991), (Michael Kalkbrener 1991), (Daniel Lazard 1991) solves this difficulty
- Moreover, for any input $F \subseteq \mathbf{k}[x_1, \ldots, x_n]$ one can compute regular chains $T_1, \ldots, T_e$ such that a point $z \in \mathbf{K}^n$ is a zero of F if and only if z is a zero of one of the $T_1, \ldots, T_e$ (in some technical sense). (Dong Ming Wang 2000), (Marc Moreno Maza 2000).

A recursive view on polynomials

Let $\mathbf{k}$ be a field, $X = x_1 < \cdots < x_n$ be variables and $f, g \in \mathbf{k}[X]$ with $g \notin \mathbf{k}$.

- $\mathrm{mvar}(g)$: the greatest variable in g is the *leader* or *main variable* of g,
- $\mathrm{init}(g)$: the leading coefficient of g w.r.t. $\mathrm{mvar}(g)$ is the *initial* of g,
- $\mathrm{mdeg}(g)$: the degree of g w.r.t. $\mathrm{mvar}(g)$,
- $\mathrm{rank}(g)$ $= v^d$ where $v = \mathrm{mvar}(g)$ and $d = \mathrm{mdeg}(g)$,
- $\mathrm{pdivide}(f, g)$ $= (q, r)$ with $q, r \in \mathbf{k}[X]$, $\deg(r, v_g) < d_g$ and $h_g^e f = qg + r$ where $h_g = \mathrm{init}(g)$, $e = \max(\deg(f, v) - d_g + 1, 0)$, $v_g = \mathrm{mvar}(g)$ and $d_g = \mathrm{mdeg}(g)$,

Example

Assume $n \geq 3$. If $p = x_1 x_3^2 - 2x_2 x_3 + 1$, then we have $\mathrm{mvar}(p) = x_3$, $\mathrm{mdeg}(p) = 2$, $\mathrm{init}(p) = x_1$ and $\mathrm{rank}(p) = x_3^2$.

Go to RegularChains.pdf Section 2.1.

Regular chain

Definition

The set $T \subset \mathbf{k}[x_n > \cdots > x_1]$ is triangular set if it consists of non-constant polynomials with pair-wise different main variables.
Define $h_T := \prod_{t \in T} \operatorname{init}(t)$, where $\operatorname{init}(t) = \operatorname{lc}(t, \operatorname{mvar}(t))$.
The *quasi-component* and *saturated ideal* of T are:

$$W(T) := V(T) \setminus V(h_T) \text{ and } \operatorname{sat}(T) = \langle T \rangle : h_T^{\infty}.$$

Regular chain

Definition

The set $T \subset \mathbf{k}[x_n > \cdots > x_1]$ is triangular set if it consists of non-constant polynomials with pair-wise different main variables.
Define $h_T := \prod_{t \in T} \mathrm{init}(t)$, where $\mathrm{init}(t) = \mathrm{lc}(t, \mathrm{mvar}(t))$.
The *quasi-component* and *saturated ideal* of T are:

$$W(T) := V(T) \setminus V(h_T) \text{ and } \mathrm{sat}(T) = \langle T \rangle : h_T^{\infty}.$$

Note that for all triangular set T we have:

- $\overline{W(T)} = V(\mathrm{sat}(T))$.
- if $\mathrm{sat}(T) \neq \langle 1 \rangle$ then $\mathrm{sat}(T)$ is strongly equi-dimensional.

Regular chain

Definition

The set $T \subset \mathbf{k}[x_n > \cdots > x_1]$ is triangular set if it consists of non-constant polynomials with pair-wise different main variables.
Define $h_T := \prod_{t \in T} \mathrm{init}(t)$, where $\mathrm{init}(t) = \mathrm{lc}(t, \mathrm{mvar}(t))$.
The *quasi-component* and *saturated ideal* of T are:

$$W(T) := V(T) \setminus V(h_T) \text{ and } \mathrm{sat}(T) = \langle T \rangle : h_T^{\infty}.$$

Note that for all triangular set T we have:

- $\overline{W(T)} = V(\mathrm{sat}(T))$.
- if $\mathrm{sat}(T) \neq \langle 1 \rangle$ then $\mathrm{sat}(T)$ is strongly equi-dimensional.

Definition (M. Kalkbrner, 1991 - L. Yang, J. Zhang 1991)

T is a *regular chain* if $T = \varnothing$ or $T := T' \cup \{t\}$ with $\mathrm{mvar}(t)$ maximum s.t.

- T' is a regular chain,
- $\mathrm{init}(t)$ is regular modulo $\mathrm{sat}(T')$.

Regular chain: alternative definition

Regular chain: alternative definition

Regular chain: algorithmic properties

Theorem (P. Aubry, D. Lazard, M., 1997)

T is a regular chain iff $\{p \mid \mathrm{prem}(p, T) = 0\} \quad = \quad \mathrm{sat}(T)$.

Regular chain: algorithmic properties

Theorem (P. Aubry, D. Lazard, M., 1997)

T *is a* regular chain *iff* $\{p \mid \mathrm{prem}(p,T) = 0\} \;=\; \mathrm{sat}(T)$.

Definition

Let $T \subset \mathbf{k}[x_n > \cdots > x_1]$ be a triangular set and $p \in \mathbf{k}[x_n > \cdots > x_1]$. If T is empty then, the *iterated resultant* of p w.r.t. T is resultant$(T,p) = p$. Otherwise, writing $T = T_{<w} \cup T_w$

$$\text{resultant}(T,p) \;=\; \begin{cases} p & \text{if } \deg(p,w) = 0 \\ \text{resultant}(T_{<w}, \text{resultant}(T_w, p, w)) & \text{otherwise} \end{cases}$$

Regular chain: algorithmic properties

Theorem (P. Aubry, D. Lazard, M., 1997)

T *is a* regular chain *iff* $\{p \mid \mathrm{prem}(p, T) = 0\} \quad = \quad \mathrm{sat}(T)$.

Definition

Let $T \subset \mathbf{k}[x_n > \cdots > x_1]$ be a triangular set and $p \in \mathbf{k}[x_n > \cdots > x_1]$. If T is empty then, the *iterated resultant* of p w.r.t. T is resultant$(T, p) = p$. Otherwise, writing $T = T_{<w} \cup T_w$

$$\text{resultant}(T, p) \quad = \quad \begin{cases} p & \text{if } \deg(p, w) = 0 \\ \text{resultant}(T_{<w}, \text{resultant}(T_w, p, w)) & \text{otherwise} \end{cases}$$

Theorem (L. Yang, J. Zhang 1991)

p *is regular modulo* $\mathrm{sat}(T)$ *iff* resultant$(T, p) \neq 0$.

Triangular decomposition of an algebraic variety

Kalkbrener triangular decomposition

Let $F \subset \mathbf{k}[\mathbf{x}]$. A family of regular chains $T_1, \ldots, T_e$ of $\mathbf{k}[\mathbf{x}]$ is called a Kalkbrener triangular decomposition of $V(F)$ if

$$V(F) \quad = \quad \cup_{i=1}^{e} V(\mathrm{sat}(T_i)).$$

Triangular decomposition of an algebraic variety

Kalkbrener triangular decomposition

Let $F \subset \mathbf{k}[\mathbf{x}]$. A family of regular chains $T_1, \ldots, T_e$ of $\mathbf{k}[\mathbf{x}]$ is called a Kalkbrener triangular decomposition of $V(F)$ if

$$V(F) \;=\; \cup_{i=1}^{e} V(\mathrm{sat}(T_i)).$$

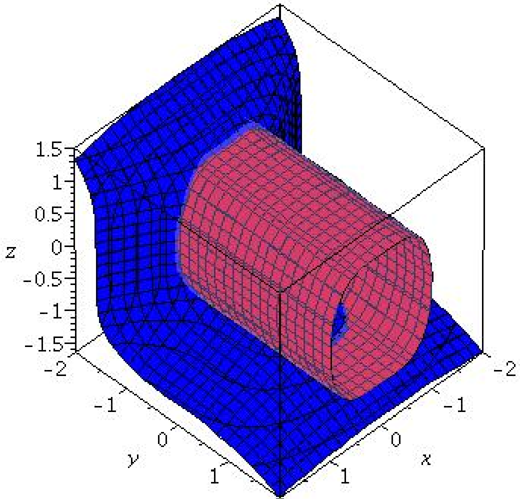
Wu-Lazard triangular decomposition

Let $F \subset \mathbf{k}[\mathbf{x}]$. A family of regular chains $T_1, \ldots, T_e$ of $\mathbf{k}[\mathbf{x}]$ is called a Wu-Lazard triangular decomposition of $V(F)$ if

$$V(F) \;=\; \cup_{i=1}^{e} W(T_i).$$

Triangularize applied to *sofa* and *cylinder* (1/2)

$$x^2 + y^3 + z^5 = x^4 + z^2 - 1 = 0$$



Triangularize applied to *sofa* and *cylinder* (2/2)

Applications Places System Sun Feb 13, 10:36 PM changbo

/home/changbo/src/maple/triade/mapleP4/lib/cylinder.mw - [Server 1] - Maple 14

File Edit View Insert Format Table Drawing Plot Spreadsheet Tools Window Help

```
> R := PolynomialRing([z, y, x]): F := [x^2+y^3+z^5, x^4+z^2-1]: dec :=
  Triangularize(F, R): map(Display, dec, R);
```

$$\left[\left\{\begin{array}{c} \left(-2\,x^4+x^8+1\right)z+x^2+y^3=0 \\ y^6+2\,x^2y^3+10\,x^{12}-10\,x^8+x^{20}-5\,x^{16}+6\,x^4-1=0 \\ -2\,x^4+x^8+1\neq 0 \end{array}\right.\right]$$

```
> dec := Triangularize(F, R, output=lazard): map(Display, dec, R);
```

$$\left[\left\{\begin{array}{c} \left(-2\,x^4+x^8+1\right)z+x^2+y^3=0 \\ y^6+2\,x^2y^3+10\,x^{12}-10\,x^8+x^{20}-5\,x^{16}+6\,x^4-1=0 \\ -2\,x^4+x^8+1\neq 0 \end{array}\right., \left\{\begin{array}{c} z=0 \\ y^2+y+1=0 \\ x^2+1=0 \end{array}\right.,\right.$$

$$\left\{\begin{array}{c} z=0 \\ y-1=0 \\ x^2+1=0 \end{array}\right., \left\{\begin{array}{c} z=0 \\ y^2-y+1=0 \\ x+1=0 \end{array}\right., \left\{\begin{array}{c} z=0 \\ y^2-y+1=0 \\ x-1=0 \end{array}\right., \left\{\begin{array}{c} z=0 \\ y+1=0 \\ x+1=0 \end{array}\right.,$$

$$\left.\left\{\begin{array}{c} z=0 \\ y+1=0 \\ x-1=0 \end{array}\right.\right]$$

Ready Memory: 0.74M Time: 0.02s Text

Relations with resultants and subresultants (1/3)

In a nutshell, solving bivariate polynomial systems can be done via

Relations with resultants and subresultants (1/3)

In a nutshell, solving bivariate polynomial systems can be done via

1. resultant computations,

Relations with resultants and subresultants (1/3)

In a nutshell, solving bivariate polynomial systems can be done via

1. resultant computations,
2. factorization of univariate polynomials, and

Relations with resultants and subresultants (1/3)

In a nutshell, solving bivariate polynomial systems can be done via

1. resultant computations,
2. factorization of univariate polynomials, and
3. univariate polynomial GCDs.

Relations with resultants and subresultants (1/3)

In a nutshell, solving bivariate polynomial systems can be done via

1. resultant computations,
2. factorization of univariate polynomials, and
3. univariate polynomial GCDs.

Example (von zur Gathen & Gerhard, Chapter 6)

Let $P = \left(y^2 + 6\right)(x - 1) - y\left(x^2 + 1\right)$ and $Q = \left(x^2 + 6\right)(y - 1) - x\left(y^2 + 1\right)$

Relations with resultants and subresultants (1/3)

In a nutshell, solving bivariate polynomial systems can be done via

1. resultant computations,
2. factorization of univariate polynomials, and
3. univariate polynomial GCDs.

Example (von zur Gathen & Gerhard, Chapter 6)

Let $P = \left(y^2+6\right)(x-1) - y\left(x^2+1\right)$ and $Q = \left(x^2+6\right)(y-1) - x\left(y^2+1\right)$

- $\mathrm{res}(\mathrm{P}, \mathrm{Q}, \mathrm{y}) = 2\left(\mathrm{x}^2 - \mathrm{x} + 4\right)(\mathrm{x}-2)^2(\mathrm{x}-3)^2$.

Relations with resultants and subresultants (1/3)

In a nutshell, solving bivariate polynomial systems can be done via

1. resultant computations,
2. factorization of univariate polynomials, and
3. univariate polynomial GCDs.

Example (von zur Gathen & Gerhard, Chapter 6)

Let $P = \left(y^2+6\right)(x-1) - y\left(x^2+1\right)$ and $Q = \left(x^2+6\right)(y-1) - x\left(y^2+1\right)$

- $\mathrm{res}(\mathrm{P},\mathrm{Q},\mathrm{y}) = 2\left(\mathrm{x}^2-\mathrm{x}+4\right)(\mathrm{x}-2)^2(\mathrm{x}-3)^2$.
- $\mathrm{gcd}(\mathrm{P},\mathrm{Q},\mathrm{x}-2=0) = (\mathrm{y}-2)(\mathrm{y}-3)$.

Relations with resultants and subresultants (1/3)

In a nutshell, solving bivariate polynomial systems can be done via

1. resultant computations,
2. factorization of univariate polynomials, and
3. univariate polynomial GCDs.

Example (von zur Gathen & Gerhard, Chapter 6)

Let $P = \left(y^2+6\right)(x-1) - y\left(x^2+1\right)$ and $Q = \left(x^2+6\right)(y-1) - x\left(y^2+1\right)$

- $\mathrm{res(P, Q, y) = 2\left(x^2 - x + 4\right)(x-2)^2(x-3)^2}$.
- $\mathrm{gcd(P, Q, x-2=0) = (y-2)(y-3)}$.
- $\mathrm{gcd(P, Q, x-3=0) = (y-2)(y-3)}$.

Relations with resultants and subresultants (1/3)

In a nutshell, solving bivariate polynomial systems can be done via

1. resultant computations,
2. factorization of univariate polynomials, and
3. univariate polynomial GCDs.

Example (von zur Gathen & Gerhard, Chapter 6)

Let $P = \left(y^2+6\right)(x-1) - y\left(x^2+1\right)$ and $Q = \left(x^2+6\right)(y-1) - x\left(y^2+1\right)$

- $\mathrm{res}(\mathrm{P}, \mathrm{Q}, \mathrm{y}) = 2\left(\mathrm{x}^2 - \mathrm{x} + 4\right)(\mathrm{x}-2)^2(\mathrm{x}-3)^2$.
- $\mathrm{gcd}(\mathrm{P}, \mathrm{Q}, \mathrm{x}-2=0) = (\mathrm{y}-2)(\mathrm{y}-3)$.
- $\mathrm{gcd}(\mathrm{P}, \mathrm{Q}, \mathrm{x}-3=0) = (\mathrm{y}-2)(\mathrm{y}-3)$.
- $\mathrm{gcd}(\mathrm{P}, \mathrm{Q}, \mathrm{x}^2-\mathrm{x}+4=0) = (2\,\mathrm{x}-1)\,\mathrm{y} - 7 - \mathrm{x}$.

Relations with resultants and subresultants (2/3)

In fact, factorizing the resultant is not necessary. Using regularity test and the specialization property of subresultants is sufficient.

Relations with resultants and subresultants (2/3)

In fact, factorizing the resultant is not necessary. Using regularity test and the specialization property of subresultants is sufficient.
Consider the following polynomials $f, g \in \mathbb{Q}[y < x]$:

$$\begin{array}{rcl} f & = & x^7 - 36x - 22y + 1, \\ g & = & x^6 + 47x^3 - 60xy^2 - 6xy - 83y^2 - 10y + 50. \end{array}$$

Relations with resultants and subresultants (2/3)

In fact, factorizing the resultant is not necessary. Using regularity test and the specialization property of subresultants is sufficient.
Consider the following polynomials $f, g \in \mathbb{Q}[y < x]$:

$$\begin{array}{rcl} f &=& x^7 - 36x - 22y + 1, \\ g &=& x^6 + 47x^3 - 60xy^2 - 6xy - 83y^2 - 10y + 50. \end{array}$$

The complete list of subresultants of (f, g) w.r.t. x is:

$$\begin{array}{rcl} S_6 &=& g, \\ S_5 &=& 56x^4 + 60x^2y^2 + 6x^2y + 83xy^2 + 10xy + 17x + 81y + 1, \\ S_4 &=& 46x^4 + 64x^2y^2 + 27x^2y + 13xy^2 + 45xy + 25x + 4y + 56, \\ S_3 &=& 74x^2y^4 + 7x^3y^2 + 56x^2y^3 + 44xy^4 + \cdots + 98y^2 + 86y + 53, \\ S_2 &=& 25x^2y^8 + 10x^2y^7 + 26xy^8 + 62x^2y^6 + \cdots + 96x + 72y + 43, \\ S_1 &=& 81xy^{12} + 28xy^{11} + 76y^{12} + 24xy^{10} + 5xy^9 + \cdots + 4x + 73y + 77, \\ S_0 &=& 97y^{15} + 82y^{14} + 82y^{13} + \cdots + 23y^5 + 89y^4 + 31y^3 + y^2 + 54y + 69. \end{array}$$

Relations with resultants and subresultants (2/3)

In fact, factorizing the resultant is not necessary. Using regularity test and the specialization property of subresultants is sufficient.
Consider the following polynomials $f, g \in \mathbb{Q}[y < x]$:

$$\begin{array}{rcl} f & = & x^7 - 36x - 22y + 1, \\ g & = & x^6 + 47x^3 - 60xy^2 - 6xy - 83y^2 - 10y + 50. \end{array}$$

The complete list of subresultants of (f, g) w.r.t. x is:

$$\begin{array}{rcl} S_6 & = & g, \\ S_5 & = & 56x^4 + 60x^2y^2 + 6x^2y + 83xy^2 + 10xy + 17x + 81y + 1, \\ S_4 & = & 46x^4 + 64x^2y^2 + 27x^2y + 13xy^2 + 45xy + 25x + 4y + 56, \\ S_3 & = & 74x^2y^4 + 7x^3y^2 + 56x^2y^3 + 44xy^4 + \cdots + 98y^2 + 86y + 53, \\ S_2 & = & 25x^2y^8 + 10x^2y^7 + 26xy^8 + 62x^2y^6 + \cdots + 96x + 72y + 43, \\ S_1 & = & 81xy^{12} + 28xy^{11} + 76y^{12} + 24xy^{10} + 5xy^9 + \cdots + 4x + 73y + 77, \\ S_0 & = & 97y^{15} + 82y^{14} + 82y^{13} + \cdots + 23y^5 + 89y^4 + 31y^3 + y^2 + 54y + 69. \end{array}$$

The solutions of $f = g = 0$ can be calculated using S_0, S_1 only.
Go to RegularChains.pdf Sections 2.2 and 2.3.

Relations with resultants and subresultants (3/3)

Extending the previous ideas to solving m polynomial equations in n variables can be done using

- a cascade of Sylvester resultants (this talk), or
- a combination of Dixon/Macaulay resultants and Sylvester resultants (work in progress).

Relations with Gröbner bases

Normalized regular chains

- The regular chain $T \subset \mathbf{k}[x_n > \cdots > x_1]$ is said *normalized* if for every $t, t' \in T$ we have $\deg(\mathrm{init}(t), \mathrm{mvar}(t')) = 0$.

Relations with Gröbner bases

Normalized regular chains

- The regular chain $T \subset \mathbf{k}[x_n > \cdots > x_1]$ is said *normalized* if for every $t, t' \in T$ we have $\deg(\mathrm{init}(t), \mathrm{mvar}(t')) = 0$.
- Let $Y := \{\mathrm{mvar}(t) \mid t \in T\}$ and $U := X \setminus Y$. If T is normalized, then T is a *Gröbner basis* of dimension 0 of the ideal it generates in $\mathbf{k}(U)[Y]$.

Relations with Gröbner bases

Normalized regular chains

- The regular chain $T \subset \mathbf{k}[x_n > \cdots > x_1]$ is said *normalized* if for every $t, t' \in T$ we have $\deg(\mathrm{init}(t), \mathrm{mvar}(t')) = 0$.
- Let $Y := \{\mathrm{mvar}(t) \mid t \in T\}$ and $U := X \setminus Y$. If T is normalized, then T is a *Gröbner basis* of dimension 0 of the ideal it generates in $\mathbf{k}(U)[Y]$.

From lexicographical Gröbner bases to regular chains

Relations with Gröbner bases

Normalized regular chains

- The regular chain $T \subset \mathbf{k}[x_n > \cdots > x_1]$ is said *normalized* if for every $t, t' \in T$ we have $\deg(\mathrm{init}(t), \mathrm{mvar}(t')) = 0$.
- Let $Y := \{\mathrm{mvar}(t) \mid t \in T\}$ and $U := X \setminus Y$. If T is normalized, then T is a *Gröbner basis* of dimension 0 of the ideal it generates in $\mathbf{k}(U)[Y]$.

From lexicographical Gröbner bases to regular chains

- Let G be a lexicographical Gröbner basis of a zero-dimensional ideal $\mathcal{I} \subset \mathbf{k}[x_n > \cdots > x_1]$. Then, Lextriangular($G$) computes regular chains (optionally normalized) $T_1, \ldots, T_e \subset \mathbf{k}[x_n > \cdots > x_1]$ so that $V(G) = \cup_{i=1}^{e} V(T_i)$. (Daniel Lazard, 1992).

Relations with Gröbner bases

Normalized regular chains

- The regular chain $T \subset \mathbf{k}[x_n > \cdots > x_1]$ is said *normalized* if for every $t, t' \in T$ we have $\deg(\mathrm{init}(t), \mathrm{mvar}(t')) = 0$.
- Let $Y := \{\mathrm{mvar}(t) \mid t \in T\}$ and $U := X \setminus Y$. If T is normalized, then T is a *Gröbner basis* of dimension 0 of the ideal it generates in $\mathbf{k}(U)[Y]$.

From lexicographical Gröbner bases to regular chains

- Let G be a lexicographical Gröbner basis of a zero-dimensional ideal $\mathcal{I} \subset \mathbf{k}[x_n > \cdots > x_1]$. Then, Lextriangular($G$) computes regular chains (optionally normalized) $T_1, \ldots, T_e \subset \mathbf{k}[x_n > \cdots > x_1]$ so that $V(G) = \cup_{i=1}^{e} V(T_i)$. (Daniel Lazard, 1992).
- This is done at a cost which is at most that inverting at most $\#G$ polynomials modulo one of the ideals $\langle T_1 \rangle, \ldots, \langle T_e \rangle$.

Relations with Gröbner bases

Normalized regular chains

- The regular chain $T \subset \mathbf{k}[x_n > \cdots > x_1]$ is said *normalized* if for every $t, t' \in T$ we have $\deg(\mathrm{init}(t), \mathrm{mvar}(t')) = 0$.
- Let $Y := \{\mathrm{mvar}(t) \mid t \in T\}$ and $U := X \setminus Y$. If T is normalized, then T is a *Gröbner basis* of dimension 0 of the ideal it generates in $\mathbf{k}(U)[Y]$.

From lexicographical Gröbner bases to regular chains

- Let G be a lexicographical Gröbner basis of a zero-dimensional ideal $\mathcal{I} \subset \mathbf{k}[x_n > \cdots > x_1]$. Then, Lextriangular$(G)$ computes regular chains (optionally normalized) $T_1, \ldots, T_e \subset \mathbf{k}[x_n > \cdots > x_1]$ so that $V(G) = \cup_{i=1}^{e} V(T_i)$. (Daniel Lazard, 1992).
- This is done at a cost which is at most that inverting at most $\#G$ polynomials modulo one of the ideals $\langle T_1 \rangle, \ldots, \langle T_e \rangle$.
- This is practically very effective.

Triangular decompositions: the incremental approach

- Let $f \in \mathbf{k}[x_1, \ldots, x_n]$ and $T \subseteq \mathbf{k}[x_1, \ldots, x_n]$ be a regular chain.

Triangular decompositions: the incremental approach

- Let $f \in \mathbf{k}[x_1, \ldots, x_n]$ and $T \subseteq \mathbf{k}[x_1, \ldots, x_n]$ be a regular chain.
- The intersection $V(f) \cap W(T)$ is approximated by the function call Intersect(f, T), which returns regular chains $T_1, \ldots, T_e \subseteq \mathbf{k}[X]$ s.t.:

$$V(f) \cap W(T) \subseteq W(T_1) \cup \cdots \cup W(T_e) \subseteq V(f) \cap \overline{W(T)},$$

where $\overline{W(T)}$ denotes the Zariski closure of $W(T)$.

Triangular decompositions: the incremental approach

- Let $f \in \mathbf{k}[x_1, \ldots, x_n]$ and $T \subseteq \mathbf{k}[x_1, \ldots, x_n]$ be a regular chain.
- The intersection $V(f) \cap W(T)$ is approximated by the function call Intersect(f, T), which returns regular chains $T_1, \ldots, T_e \subseteq \mathbf{k}[X]$ s.t.:

$$V(f) \cap W(T) \subseteq W(T_1) \cup \cdots \cup W(T_e) \subseteq V(f) \cap \overline{W(T)},$$

  where $\overline{W(T)}$ denotes the Zariski closure of $W(T)$.
- Given $f_1, \ldots, f_m \in \mathbf{k}[x_1, \ldots, x_n]$, one can solve $f_1 = \cdots = f_m = 0$ using repeated calls to Intersect.

Triangular decompositions: the incremental approach

- Let $f \in \mathbf{k}[x_1, \ldots, x_n]$ and $T \subseteq \mathbf{k}[x_1, \ldots, x_n]$ be a regular chain.
- The intersection $V(f) \cap W(T)$ is approximated by the function call $\mathsf{Intersect}(f, T)$, which returns regular chains $T_1, \ldots, T_e \subseteq \mathbf{k}[X]$ s.t.:

$$V(f) \cap W(T) \subseteq W(T_1) \cup \cdots \cup W(T_e) \subseteq V(f) \cap \overline{W(T)},$$

  where $\overline{W(T)}$ denotes the Zariski closure of $W(T)$.
- Given $f_1, \ldots, f_m \in \mathbf{k}[x_1, \ldots, x_n]$, one can solve $f_1 = \cdots = f_m = 0$ using repeated calls to Intersect.
- Indeed, if $V(f_1, \ldots, f_{m-1}) = \cup_{i=1}^{e} W(T_i)$, then we have

$$V(f_1, \ldots, f_m) = \cup_{i=1}^{e} \, \mathsf{Intersect}(f_m, T_i).$$

(Daniel Lazard 1991), (M. 2000), (Changbo Chen & M. 2011-2012).

Outline

1. Triangular decompositions in polynomial system solving

2. Modular methods in polynomial system solving

3. A Modular methods for incremental triangular decompositions

4. Conclusions

Computing by homomotphic images: principles,

Examples

- The computation of the determinant of an integer matrix using the Chinese Remaindering Theorem.

Computing by homomotphic images: principles,

Examples

- The computation of the determinant of an integer matrix using the Chinese Remaindering Theorem.
- Since resultants are determinants, similar strategies apply.

Computing by homomotphic images: principles,

Examples

- The computation of the determinant of an integer matrix using the Chinese Remaindering Theorem.
- Since resultants are determinants, similar strategies apply.
- A more advanced example is the computation of the GCD of univariate integer polynomials, again using CRT.

Computing by homomotphic images: principles,

Examples

- The computation of the determinant of an integer matrix using the Chinese Remaindering Theorem.
- Since resultants are determinants, similar strategies apply.
- A more advanced example is the computation of the GCD of univariate integer polynomials, again using CRT.
- See the book by von zur Gathen & Gerhard.

Computing by homomotphic images: principles,

Examples

- The computation of the determinant of an integer matrix using the Chinese Remaindering Theorem.
- Since resultants are determinants, similar strategies apply.
- A more advanced example is the computation of the GCD of univariate integer polynomials, again using CRT.
- See the book by von zur Gathen & Gerhard.

Advantages and issues

- Modular methods (1) may control expression swell, (2) allow sharper implementation (fine control memory), (3) open the door to FFT-based arithmetic, and (4) provide opportunities for concurrency.

Computing by homomotphic images: principles,

Examples

- The computation of the determinant of an integer matrix using the Chinese Remaindering Theorem.
- Since resultants are determinants, similar strategies apply.
- A more advanced example is the computation of the GCD of univariate integer polynomials, again using CRT.
- See the book by von zur Gathen & Gerhard.

Advantages and issues

- Modular methods (1) may control expression swell, (2) allow sharper implementation (fine control memory), (3) open the door to FFT-based arithmetic, and (4) provide opportunities for concurrency.
- Modular methods are (1) generally harder to implement than direct methods, and (2) usually require change of representations which may come with significant costs in terms of memory consumption.

Expression swell may sometimes be handled in other ways

- Consider the system F (Barry Trager).

$$-x^5 + y^5 - 3y - 1 = 5y^4 - 3 = -20x + y - z = 0$$

We solve it for $z < y < x$.

- $V(F)$ is equiprojectable and its Lazard triangular set is
  - [illegible]
  - [illegible]
  - $3125z^{20} - 9375z^{16} - 40000000000z^{15} - 2015999988750z^{12} - 1560000000000z^{11} + 192000000000000000z^{10} - 121651253568000006750z^{8} - 1474560223200000000z^{7} - 6528000000000000000z^{6} - 4096000000000000000000000z^{5} - 16986908639233347839997975z^{4} - 141557671526403024000000000z^{3} - 589823873280000000000000z^{2} - 12288000000000000000000000z - 619530361923198287873244160024 3$
- Applying the transformation of Dahan and Schost leads to 1787 characters.
  - $(20z^{19} + (-48z^{15}) + (-192000000z^{14}) + (-(38707199784/5)z^{11}) + (-5491200000z^{10}) + 614400000000000z^{9} + (-(7785680228352004 32/25)z^{7}) + (-3303014899968000 0z^{6}) + (-125337600000000000z^{5}) + (-65536000000000000000z^{4}) + (-(27179053822773356543996 76/125)z^{3}) + (-1358953646653469030 4000z^{2}) + (-377487278899200000000z) - 3932160000000000000 00)x + 3200000z^{15} + 161280000z^{12} + 124800000z^{11} + (-30720000000000z^{10}) + 194641962854400 0z^{8} + 2359296178560000z^{7} + 10444800000000000z^{6} + 98304000000000000000z^{5} + 407685987827722782720 0z^{4} + 33973848244224241920 00z^{3} + 1415577397248000000000z^{2} + 294912000000000000000z + 198249699507965678059619 5328$
  - $(20z^{19} + (-48z^{15}) + (-192000000z^{14}) + (-(38707199784/5)z^{11}) + (-5491200000z^{10}) + 614400000000000z^{9} + (-(778568022835200432/25)z^{7}) + (-33030148999680000z^{6}) + (-125337600000000000z^{5}) + (-65536000000000000000z^{4}) + (-(2717905382277335654399676/125)z^{3}) + (-13589536466534690304000z^{2}) + (-377487278899200000000z) - 393216000000000000000)y + (-12z^16) + (-(9676799856/5)z^{12}) + (-1996800000z^{11}) + (-(194642219980800648/25)z^{8}) + (-141557817139200 00z^{7}) + (-835584000000000z^{6}) + (-(679471833416273049598704/125)z^{4}) + (-905967682191476121600 0z^{3}) + (-566230715596800000000z^{2}) + (-157286400000000000000z) + (-2038432221757477324800972/625)$
  - $z^20 + (-3z^16) + (-12800000z^{15}) + (-(3225599982/5)z^{12}) + (-499200000z^{11}) + 61440000000000z^{10} + (-(97321002854400054/25)z^{8}) + (-4718592714240000z^{7}) + (-2088960000000000z^{6}) + (-131072000000000000000z^{5}) + (-(6794763455693339135999 19/125)z^{4}) + (-4529845488844896768000z^{3}) + (-188743639449600000000z^{2}) + (-393216000000000000000z) + (-619530361923198287873244160024 3/3125)$
- One can do better! Here's the regular chain produced by the `Triangularize` algorithm of the `RegularChains` library, counting 963 characters.
  - $20x - 1y + z$
  - $\left((4375z^{12} + 52800011625z^{8} + 32000000000z^{7} + 110591902080002925z^{4} + 614399808000000000z^{3} + 128000000000000000z^{2} + 566231172710410368000 27\right)y - 1875z^{13} - 9600010125z^{9} + 2000000000z^{8} - 73727147520045 45z^{5} + 307200024000000000z^{4} + 128000000000000000z^{3} - 22118403456000135z + 235929636864001440000 00$
  - $3125z^{20} - 9375z^{16} - 40000000000z^{15} - 2015999988750z^{12} - 1560000000000z^{11} + 192000000000000000z^{10} - 121651253568000006750z^{8} - 1474560223200000000z^{7} - 6528000000000000000z^{6} - 4096000000000000000000000z^{5} - 16986908639233347839997975z^{4} - 141557671526403024000000000z^{3} - 589823873280000000000000z^{2} - 12288000000000000000000000z - 6195303619231982878732441600243$

Trace algorithms

- Consider an algorithm Solver(F) taking $F \subseteq \mathbb{Z}[x_n > \cdots > x_1]$ computing a finite sequence $\mathcal{G}$ of finite sets $G_1, G_2, \ldots, \subseteq \langle F \rangle$ until $G_i = G_{\text{output}}$ satisifies a property, e.g. Gröbner basis of $\langle F \rangle$ or Wu-characteristic set of F.

Trace algorithms

- Consider an algorithm Solver(F) taking $F \subseteq \mathbb{Z}[x_n > \cdots > x_1]$ computing a finite sequence $\mathcal{G}$ of finite sets $G_1, G_2, \ldots, \subseteq \langle F \rangle$ until $G_i = G_{\mathrm{output}}$ satisifies a property, e.g. Gröbner basis of $\langle F \rangle$ or Wu-characteristic set of F.
- Endow all such finite sequences $\mathcal{G}$ with a *rank function* so that, for every well-chosen prime number p, the sequence computed by Solver$(F \bmod p)$ has maximum rank iff Solver$(F \bmod p) = G_{\mathrm{output}} \bmod p$.

Trace algorithms

- Consider an algorithm Solver(F) taking $F \subseteq \mathbb{Z}[x_n > \cdots > x_1]$ computing a finite sequence $\mathcal{G}$ of finite sets $G_1, G_2, \ldots, \subseteq \langle F \rangle$ until $G_i = G_{\mathrm{output}}$ satisifies a property, e.g. Gröbner basis of $\langle F \rangle$ or Wu-characteristic set of F.
- Endow all such finite sequences $\mathcal{G}$ with a *rank function* so that, for every well-chosen prime number p, the sequence computed by Solver($F \bmod p$) has maximum rank iff Solver($F \bmod p$) $= G_{\mathrm{output}} \bmod p$.
- For polynomial GCD computations (with $n = 1$) one can simply use the degree as rank function.

Trace algorithms

- Consider an algorithm Solver(F) taking $F \subseteq \mathbb{Z}[x_n > \cdots > x_1]$ computing a finite sequence $\mathcal{G}$ of finite sets $G_1, G_2, \ldots, \subseteq \langle F \rangle$ until $G_i = G_{\text{output}}$ satisifies a property, e.g. Gröbner basis of $\langle F \rangle$ or Wu-characteristic set of F.
- Endow all such finite sequences $\mathcal{G}$ with a *rank function* so that, for every well-chosen prime number p, the sequence computed by Solver($F \bmod p$) has maximum rank iff Solver($F \bmod p$) $= G_{\text{output}} \bmod p$.
- For polynomial GCD computations (with $n = 1$) one can simply use the degree as rank function.
- For Gröbner bases, one can use the Hilbert function (Carlo Traverso, ISSAC 1988), (Jean-Charles Faugère, PASCO 1994), (Elizabeth Arnold, JSC 2003)

Trace algorithms

- Consider an algorithm Solver(F) taking $F \subseteq \mathbb{Z}[x_n > \cdots > x_1]$ computing a finite sequence $\mathcal{G}$ of finite sets $G_1, G_2, \ldots, \subseteq \langle F \rangle$ until $G_i = G_{\mathrm{output}}$ satisifies a property, e.g. Gröbner basis of $\langle F \rangle$ or Wu-characteristic set of F.
- Endow all such finite sequences $\mathcal{G}$ with a *rank function* so that, for every well-chosen prime number p, the sequence computed by Solver($F \bmod p$) has maximum rank iff Solver($F \bmod p$) $= G_{\mathrm{output}} \bmod p$.
- For polynomial GCD computations (with $n = 1$) one can simply use the degree as rank function.
- For Gröbner bases, one can use the Hilbert function (Carlo Traverso, ISSAC 1988), (Jean-Charles Faugère, PASCO 1994), (Elizabeth Arnold, JSC 2003)
- For characteristic sets, one can use the notion of rank as defined by Ritt and Wu (M. ACA 2003).

The case of decomposition algorithms

- Consider an algorithm Solver(F) taking $F \subseteq \mathbb{Q}[x_n > \cdots > x_1]$ (assumed to be zero-dimensional for simplicity) computing a triangular decomposition into regular chain $T_1, \ldots, T_e$.

The case of decomposition algorithms

- Consider an algorithm Solver(F) taking $F \subseteq \mathbb{Q}[x_n > \cdots > x_1]$ (assumed to be zero-dimensional for simplicity) computing a triangular decomposition into regular chain $T_1, \ldots, T_e$.
- The algorithm `EquiprojectableDecompositon`$(T_1, \ldots, T_e)$ returns a canonical triangular decomposition of $V(F)$ based on "geometrical" considerations.

The case of decomposition algorithms

- Consider an algorithm Solver(F) taking $F \subseteq \mathbb{Q}[x_n > \cdots > x_1]$ (assumed to be zero-dimensional for simplicity) computing a triangular decomposition into regular chain $T_1, \ldots, T_e$.
- The algorithm `EquiprojectableDecompositon`$(T_1, \ldots, T_e)$ returns a canonical triangular decomposition of $V(F)$ based on "geometrical" considerations.
- Moreover, if the prime p is large enough, then the decompositions `EquiprojectableDecompositon`(Solver(F $\mod p$)) and `EquiprojectableDecompositon`(Solver(F)) $\mod p$ match.

The case of decomposition algorithms

- Consider an algorithm Solver(F) taking $F \subseteq \mathbb{Q}[x_n > \cdots > x_1]$ (assumed to be zero-dimensional for simplicity) computing a triangular decomposition into regular chain $T_1, \ldots, T_e$.
- The algorithm `EquiprojectableDecompositon`($T_1, \ldots, T_e$) returns a canonical triangular decomposition of $V(F)$ based on "geometrical" considerations.
- Moreover, if the prime p is large enough, then the decompositions `EquiprojectableDecompositon`(Solver(F $\mod p$)) and `EquiprojectableDecompositon`(Solver(F)) $\mod p$ match.
- Using Hensel lifting techniques (Schost 2002) this leads to an effective modular method for Solver(F) (Dahan, M., Schost, Wu & Xie ISSAC 2005).

Issues with iterated subresultant chains (1/2)

- Testing regularity of $p \in \mathbf{k}[x_n > \cdots > x_1]$ w.r.t. regular chain $T \subset \mathbf{k}[x_n > \cdots > x_1]$ is equivalent to checking whether resultant(T, p) is zero or not.

Issues with iterated subresultant chains (1/2)

- Testing regularity of $p \in \mathbf{k}[x_n > \cdots > x_1]$ w.r.t. regular chain $T \subset \mathbf{k}[x_n > \cdots > x_1]$ is equivalent to checking whether resultant(T, p) is zero or not.
- Moreover, elimninating variables with pseudo-division (or variants) leads to computing cascade of (pseudo-)remainder sequences and thus (multiples of) iterated resultants.

Issues with iterated subresultant chains (1/2)

- Testing regularity of $p \in \mathbf{k}[x_n > \cdots > x_1]$ w.r.t. regular chain $T \subset \mathbf{k}[x_n > \cdots > x_1]$ is equivalent to checking whether resultant(T, p) is zero or not.
- Moreover, elimnating variables with pseudo-division (or variants) leads to computing cascade of (pseudo-)remainder sequences and thus (multiples of) iterated resultants.
- Those iterated resultants usually contain large extraneous factors.

Issues with iterated subresultant chains (1/2)

- Testing regularity of $p \in \mathbf{k}[x_n > \cdots > x_1]$ w.r.t. regular chain $T \subset \mathbf{k}[x_n > \cdots > x_1]$ is equivalent to checking whether resultant(T, p) is zero or not.
- Moreover, elimnating variables with pseudo-division (or variants) leads to computing cascade of (pseudo-)remainder sequences and thus (multiples of) iterated resultants.
- Those iterated resultants usually contain large extraneous factors.
- In (C. Chen & M. JSC 2012) we give examples of 3 zero-dimensional systems with $4^3 = 64$ solutions where the extraneous factors have degree in the 1000's.

Issues with iterated subresultant chains (2/2)

- Let $C = \{t_1, t_2, \ldots, t_n\}$ be a zero-dimensional regular chain and f be a polynomial all in $\mathbf{k}[X]$.

Issues with iterated subresultant chains (2/2)

- Let $C = \{t_1, t_2, \ldots, t_n\}$ be a zero-dimensional regular chain and f be a polynomial all in $\mathbf{k}[X]$.
- For $i = 1, \ldots, n$, we denote by h_i, the initial of t_i,

Issues with iterated subresultant chains (2/2)

- Let $C = \{t_1, t_2, \ldots, t_n\}$ be a zero-dimensional regular chain and f be a polynomial all in $\mathbf{k}[X]$.
- For $i = 1, \ldots, n$, we denote by h_i, the initial of t_i,
- For $i = 1, \ldots, n-1$, we define $f_i = \mathrm{res}(\{t_{i+1}, \ldots, t_n\}, f)$ and $e_i = \deg(f_i, x_i)$, with $e_n = \deg(f, X_n)$.

Issues with iterated subresultant chains (2/2)

- Let $C = \{t_1, t_2, \ldots, t_n\}$ be a zero-dimensional regular chain and f be a polynomial all in $\mathbf{k}[X]$.
- For $i = 1, \ldots, n$, we denote by h_i, the initial of t_i,
- For $i = 1, \ldots, n-1$, we define $f_i = \mathrm{res}(\{t_{i+1}, \ldots, t_n\}, f)$ and $e_i = \deg(f_i, x_i)$, with $e_n = \deg(f, X_n)$.
- Then, the iterated resultant $\mathrm{res}(C, f)$ is given by:

$$h_1^{e_1}\left(\prod_{\beta_1 \in V_M(t_1)} h_2(\beta_1)\right)^{e_2} \cdots \left(\prod_{\beta_{n-1} \in V_M(t_1,\ldots,t_{n-1})} h_n(\beta_{n-1})\right)^{e_n} R(C, f),$$

where

$$R(C, f) = \left(\prod_{\alpha \in V_M(C)} f(\alpha)\right),$$

and $V_M(C)$ is the set of the zeros of C counted with multiplicity.

Issues with iterated subresultant chains (2/2)

- Let $C = \{t_1, t_2, \ldots, t_n\}$ be a zero-dimensional regular chain and f be a polynomial all in $\mathbf{k}[X]$.
- For $i = 1, \ldots, n$, we denote by h_i, the initial of t_i,
- For $i = 1, \ldots, n-1$, we define $f_i = \mathrm{res}(\{t_{i+1}, \ldots, t_n\}, f)$ and $e_i = \deg(f_i, x_i)$, with $e_n = \deg(f, X_n)$.
- Then, the iterated resultant $\mathrm{res}(C, f)$ is given by:

$$h_1^{e_1} \left(\prod_{\beta_1 \in V_M(t_1)} h_2(\beta_1) \right)^{e_2} \cdots \left(\prod_{\beta_{n-1} \in V_M(t_1, \ldots, t_{n-1})} h_n(\beta_{n-1}) \right)^{e_n} R(C, f),$$

where

$$R(C, f) = \left(\prod_{\alpha \in V_M(C)} f(\alpha) \right),$$

and $V_M(C)$ is the set of the zeros of C counted with multiplicity.
- Thus, if $h_1 = \cdots = h_n = 1$, then we simply have:

$$\mathrm{res}(C, f) = R(C, f).$$

Outline

1. Triangular decompositions in polynomial system solving
2. Modular methods in polynomial system solving
3. A Modular methods for incremental triangular decompositions
4. Conclusions

Recall the incremental approach and define our goals

- Let $f \in \mathbb{Q}[X_1, \ldots, X_n]$ and $T \subseteq \mathbb{Q}[X_1, \ldots, X_n]$ be a regular chain.

Recall the incremental approach and define our goals

- Let $f \in \mathbb{Q}[X_1, \ldots, X_n]$ and $T \subseteq \mathbb{Q}[X_1, \ldots, X_n]$ be a regular chain.
- The intersection $V(f) \cap W(T)$ is approximated by the function call Intersect(f, T), which returns regular chains $T_1, \ldots, T_e \subseteq \mathbf{k}[X]$ s.t.:

$$V(f) \cap W(T) \subseteq W(T_1) \cup \cdots \cup W(T_e) \subseteq V(f) \cap \overline{W(T)}.$$

Recall the incremental approach and define our goals

- Let $f \in \mathbb{Q}[X_1, \ldots, X_n]$ and $T \subseteq \mathbb{Q}[X_1, \ldots, X_n]$ be a regular chain.
- The intersection $V(f) \cap W(T)$ is approximated by the function call Intersect(f, T), which returns regular chains $T_1, \ldots, T_e \subseteq \mathbf{k}[X]$ s.t.:

$$V(f) \cap W(T) \subseteq W(T_1) \cup \cdots \cup W(T_e) \subseteq V(f) \cap \overline{W(T)}.$$

- Our goals
  1. compute modulo a well-chosen prime as in (Dahan et al., ISSAC 2005)
  2. reduce to the case where T is zero-dimensional and normalized, by variable specialization
  3. recover the specialized variables, then the rational coefficients.

Recall the incremental approach and define our goals

- Let $f \in \mathbb{Q}[X_1, \ldots, X_n]$ and $T \subseteq \mathbb{Q}[X_1, \ldots, X_n]$ be a regular chain.
- The intersection $V(f) \cap W(T)$ is approximated by the function call Intersect(f, T), which returns regular chains $T_1, \ldots, T_e \subseteq \mathbf{k}[X]$ s.t.:

$$V(f) \cap W(T) \subseteq W(T_1) \cup \cdots \cup W(T_e) \subseteq V(f) \cap \overline{W(T)}.$$

- Our goals
  1. compute modulo a well-chosen prime as in (Dahan et al., ISSAC 2005)
  2. reduce to the case where T is zero-dimensional and normalized, by variable specialization
  3. recover the specialized variables, then the rational coefficients.
- We want to avoid the recourse to Gröbner bases so as to support:
  1. algorithms in differential algebra, and
  2. positive-dimensional systems for which methods based on regular chains may have smaller output.

Recall the incremental approach and define our goals

- Let $f \in \mathbb{Q}[X_1, \ldots, X_n]$ and $T \subseteq \mathbb{Q}[X_1, \ldots, X_n]$ be a regular chain.
- The intersection $V(f) \cap W(T)$ is approximated by the function call $\mathsf{Intersect}(f, T)$, which returns regular chains $T_1, \ldots, T_e \subseteq \mathbf{k}[X]$ s.t.:

$$V(f) \cap W(T) \subseteq W(T_1) \cup \cdots \cup W(T_e) \subseteq V(f) \cap \overline{W(T)}.$$

- Our goals
  1. compute modulo a well-chosen prime as in (Dahan et al., ISSAC 2005)
  2. reduce to the case where T is zero-dimensional and normalized, by variable specialization
  3. recover the specialized variables, then the rational coefficients.
- We want to avoid the recourse to Gröbner bases so as to support:
  1. algorithms in differential algebra, and
  2. positive-dimensional systems for which methods based on regular chains may have smaller output.
- This is similar in spirit to (Grégoire Lecerf, J. Complex 2001)

Recall the incremental approach and define our goals

- Let $f \in \mathbb{Q}[X_1, \ldots, X_n]$ and $T \subseteq \mathbb{Q}[X_1, \ldots, X_n]$ be a regular chain.
- The intersection $V(f) \cap W(T)$ is approximated by the function call Intersect(f, T), which returns regular chains $T_1, \ldots, T_e \subseteq \mathbf{k}[X]$ s.t.:

$$V(f) \cap W(T) \subseteq W(T_1) \cup \cdots \cup W(T_e) \subseteq V(f) \cap \overline{W(T)}.$$

- Our goals
  1. compute modulo a well-chosen prime as in (Dahan et al., ISSAC 2005)
  2. reduce to the case where T is zero-dimensional and normalized, by variable specialization
  3. recover the specialized variables, then the rational coefficients.
- We want to avoid the recourse to Gröbner bases so as to support:
  1. algorithms in differential algebra, and
  2. positive-dimensional systems for which methods based on regular chains may have smaller output.
- This is similar in spirit to (Grégoire Lecerf, J. Complex 2001)
- However, we avoid random changes of coordinates and support decompositions in the sense of Lazard.

Notations

- Let $f, t_2, \ldots, t_n \in \mathbf{k}[X_1 < \cdots < X_n]$ be non-constant.

Notations

- Let $f, t_2, \ldots, t_n \in \mathbf{k}[X_1 < \cdots < X_n]$ be non-constant.
- Assume $T := \{t_2, \ldots, t_n\}$ is a regular chain with $\mathrm{mvar}(t_i) = X_i$.

Notations

- Let $f, t_2, \ldots, t_n \in \mathbf{k}[X_1 < \cdots < X_n]$ be non-constant.
- Assume $T := \{t_2, \ldots, t_n\}$ is a regular chain with $\mathrm{mvar}(t_i) = X_i$.
- Assume $\mathrm{mvar}(f) = X_n$.

Notations

- Let $f, t_2, \ldots, t_n \in \mathbf{k}[X_1 < \cdots < X_n]$ be non-constant.
- Assume $T := \{t_2, \ldots, t_n\}$ is a regular chain with $\mathrm{mvar}(t_i) = X_i$.
- Assume $\mathrm{mvar}(f) = X_n$.
- For convenience, we define $r_n := f$.

Notations

- Let $f, t_2, \ldots, t_n \in \mathbf{k}[X_1 < \cdots < X_n]$ be non-constant.
- Assume $T := \{t_2, \ldots, t_n\}$ is a regular chain with $\mathrm{mvar}(t_i) = X_i$.
- Assume $\mathrm{mvar}(f) = X_n$.
- For convenience, we define $r_n := f$.
- Let $S(t_n, r_n, X_n)$ be the subresultant chain of t_n and r_n regarded as polynomials in $(\mathbf{k}[X_1, \ldots, X_{n-1}])[X_n]$.

Notations

- Let $f, t_2, \ldots, t_n \in \mathbf{k}[X_1 < \cdots < X_n]$ be non-constant.
- Assume $T := \{t_2, \ldots, t_n\}$ is a regular chain with $\mathrm{mvar}(t_i) = X_i$.
- Assume $\mathrm{mvar}(f) = X_n$.
- For convenience, we define $r_n := f$.
- Let $S(t_n, r_n, X_n)$ be the subresultant chain of t_n and r_n regarded as polynomials in $(\mathbf{k}[X_1, \ldots, X_{n-1}])[X_n]$.
- Let r_{n-1} and g_n be the subresultants of index 0 and 1 from $S(t_n, r_n, X_n)$.

Notations

- Let $f, t_2, \ldots, t_n \in \mathbf{k}[X_1 < \cdots < X_n]$ be non-constant.
- Assume $T := \{t_2, \ldots, t_n\}$ is a regular chain with $\mathrm{mvar}(t_i) = X_i$.
- Assume $\mathrm{mvar}(f) = X_n$.
- For convenience, we define $r_n := f$.
- Let $S(t_n, r_n, X_n)$ be the subresultant chain of t_n and r_n regarded as polynomials in $(\mathbf{k}[X_1, \ldots, X_{n-1}])[X_n]$.
- Let r_{n-1} and g_n be the subresultants of index 0 and 1 from $S(t_n, r_n, X_n)$.
- For $2 \le i \le n-1$, let $S(t_i, r_i, X_i)$ the subresultant chain of t_i and r_i regarded as polynomials in $(\mathbf{k}[X_1, \ldots, X_{i-1}])[X_i]$.

Notations

- Let $f, t_2, \ldots, t_n \in \mathbf{k}[X_1 < \cdots < X_n]$ be non-constant.
- Assume $T := \{t_2, \ldots, t_n\}$ is a regular chain with $\mathrm{mvar}(t_i) = X_i$.
- Assume $\mathrm{mvar}(f) = X_n$.
- For convenience, we define $r_n := f$.
- Let $S(t_n, r_n, X_n)$ be the subresultant chain of t_n and r_n regarded as polynomials in $(\mathbf{k}[X_1, \ldots, X_{n-1}])[X_n]$.
- Let r_{n-1} and g_n be the subresultants of index 0 and 1 from $S(t_n, r_n, X_n)$.
- For $2 \le i \le n-1$, let $S(t_i, r_i, X_i)$ the subresultant chain of t_i and r_i regarded as polynomials in $(\mathbf{k}[X_1, \ldots, X_{i-1}])[X_i]$.
- Let r_{i-1} and g_i be the subresultants of index 0 and 1 from $S(t_i, r_i, X_i)$.

Notations

- Let $f, t_2, \ldots, t_n \in \mathbf{k}[X_1 < \cdots < X_n]$ be non-constant.
- Assume $T := \{t_2, \ldots, t_n\}$ is a regular chain with $\mathrm{mvar}(t_i) = X_i$.
- Assume $\mathrm{mvar}(f) = X_n$.
- For convenience, we define $r_n := f$.
- Let $S(t_n, r_n, X_n)$ be the subresultant chain of t_n and r_n regarded as polynomials in $(\mathbf{k}[X_1, \ldots, X_{n-1}])[X_n]$.
- Let r_{n-1} and g_n be the subresultants of index 0 and 1 from $S(t_n, r_n, X_n)$.
- For $2 \leq i \leq n-1$, let $S(t_i, r_i, X_i)$ the subresultant chain of t_i and r_i regarded as polynomials in $(\mathbf{k}[X_1, \ldots, X_{i-1}])[X_i]$.
- Let r_{i-1} and g_i be the subresultants of index 0 and 1 from $S(t_i, r_i, X_i)$.
- We denote by $\overline{s}$ the squarefree part of $s := r_1$.

Base result

- H1 for $1 \le i \le n-1$, we have $r_i \notin \mathbf{k}$ and $\mathrm{mvar}(r_i) = X_i$,
- H2 For $2 \le i \le n$, we have $g_i \notin \mathbf{k}$ and $\mathrm{mvar}(g_i) = X_i$,
- H3 The polynomial set $C := \{\overline{s}, g_2, \ldots g_n\}$ is a regular chain,
- H4 For every $2 \le i \le n$, $\mathrm{lc}(t_i, X_i)$ is invertible modulo $\langle \overline{s}, g_2, \ldots, g_{i-1} \rangle$.

Base result

- H1 for $1 \le i \le n-1$, we have $r_i \notin \mathbf{k}$ and $\mathrm{mvar}(r_i) = X_i$,
- H2 For $2 \le i \le n$, we have $g_i \notin \mathbf{k}$ and $\mathrm{mvar}(g_i) = X_i$,
- H3 The polynomial set $C := \{\overline{s}, g_2, \ldots g_n\}$ is a regular chain,
- H4 For every $2 \le i \le n$, $\mathrm{lc}(t_i, X_i)$ is invertible modulo $\langle \overline{s}, g_2, \ldots, g_{i-1} \rangle$.

Theorem

With our four Hypotheses, we have:

$$V(f, t_2, \ldots, t_n) = V(\overline{s}, g_2, \ldots, g_n).$$

Base result

H1 for $1 \le i \le n-1$, we have $r_i \notin \mathbf{k}$ and $\mathrm{mvar}(r_i) = X_i$,

H2 For $2 \le i \le n$, we have $g_i \notin \mathbf{k}$ and $\mathrm{mvar}(g_i) = X_i$,

H3 The polynomial set $C := \{\overline{s}, g_2, \ldots g_n\}$ is a regular chain,

H4 For every $2 \le i \le n$, $\mathrm{lc}(t_i, X_i)$ is invertible modulo $\langle \overline{s}, g_2, \ldots, g_{i-1} \rangle$.

Theorem

With our four Hypotheses, we have:

$$V(f, t_2, \ldots, t_n) = V(\overline{s}, g_2, \ldots, g_n).$$

- Under Hypotheses 1, 2, 3 and 4, the regular chain C can be computed by a cascade of subresultant chain computations.

Base result

H1 for $1 \le i \le n-1$, we have $r_i \notin \mathbf{k}$ and $\mathrm{mvar}(r_i) = X_i$,

H2 For $2 \le i \le n$, we have $g_i \notin \mathbf{k}$ and $\mathrm{mvar}(g_i) = X_i$,

H3 The polynomial set $C := \{\overline{s}, g_2, \ldots g_n\}$ is a regular chain,

H4 For every $2 \le i \le n$, $\mathrm{lc}(t_i, X_i)$ is invertible modulo $\langle \overline{s}, g_2, \ldots, g_{i-1} \rangle$.

Theorem

With our four Hypotheses, we have:

$$V(f, t_2, \ldots, t_n) = V(\overline{s}, g_2, \ldots, g_n).$$

- Under Hypotheses 1, 2, 3 and 4, the regular chain C can be computed by a cascade of subresultant chain computations.
- C can be computed in a modular fashion (as we see shortly) improving performance w.r.t. the direct and non-modular approach.

Base result

- H1 for $1 \le i \le n-1$, we have $r_i \notin \mathbf{k}$ and $\mathrm{mvar}(r_i) = X_i$,
- H2 For $2 \le i \le n$, we have $g_i \notin \mathbf{k}$ and $\mathrm{mvar}(g_i) = X_i$,
- H3 The polynomial set $C := \{\overline{s}, g_2, \ldots g_n\}$ is a regular chain,
- H4 For every $2 \le i \le n$, $\mathrm{lc}(t_i, X_i)$ is invertible modulo $\langle \overline{s}, g_2, \ldots, g_{i-1} \rangle$.

Theorem

With our four Hypotheses, we have:

$$V(f, t_2, \ldots, t_n) = V(\overline{s}, g_2, \ldots, g_n).$$

- Under Hypotheses 1, 2, 3 and 4, the regular chain C can be computed by a cascade of subresultant chain computations.
- C can be computed in a modular fashion (as we see shortly) improving performance w.r.t. the direct and non-modular approach.
- This modular method can be enhanced so that the 4 Hypotheses are no longer necessary (as we will see later).

$R := PolynomialRing([x3,\ x2,\ x1]):$

$f := (x2 + x1) \cdot x3^2 + x3 + 1;$

$$f := (x2 + x1)\, x3^2 + x3 + 1 \tag{1}$$

$t3 := x1 \cdot x3^2 + x2 \cdot x3 + 1;$

$$t3 := x1\, x3^2 + x2\, x3 + 1 \tag{2}$$

$t2 := (x1 + 1) \cdot x2^2 + x2 + 2;$

$$t2 := (x1 + 1)\, x2^2 + x2 + 2 \tag{3}$$

$R := PolynomialRing([x3, x2, x1]):$
$f := (x2 + x1) \cdot x3^2 + x3 + 1;$

$$f := (x2 + x1)\, x3^2 + x3 + 1 \tag{1}$$

$t3 := x1 \cdot x3^2 + x2 \cdot x3 + 1;$

$$t3 := x1\, x3^2 + x2\, x3 + 1 \tag{2}$$

$t2 := (x1 + 1) \cdot x2^2 + x2 + 2;$

$$t2 := (x1 + 1)\, x2^2 + x2 + 2 \tag{3}$$

$src1 := SubresultantChain(f, t3, x3, R):$
$g3 := SubresultantOfIndex(1, src1, R); r := SubresultantOfIndex(0, src1, R);$

$$g3 := x1\, x2\, x3 + x2^2\, x3 - x1\, x3 + x2$$
$$r := x1\, x2^2 + x2^3 - 2\, x1\, x2 + x1 \tag{4}$$

$R := PolynomialRing([x3,\ x2,\ x1]):$
$f := (x2 + x1) \cdot x3^2 + x3 + 1;$

$$f := (x2 + x1)\, x3^2 + x3 + 1 \tag{1}$$

$t3 := x1 \cdot x3^2 + x2 \cdot x3 + 1;$

$$t3 := x1\, x3^2 + x2\, x3 + 1 \tag{2}$$

$t2 := (x1 + 1) \cdot x2^2 + x2 + 2;$

$$t2 := (x1 + 1)\, x2^2 + x2 + 2 \tag{3}$$

$src1 := SubresultantChain(f, t3, x3, R):$
$g3 := SubresultantOfIndex(1, src1, R);\ r := SubresultantOfIndex(0, src1, R);$

$$g3 := x1\, x2\, x3 + x2^2\, x3 - x1\, x3 + x2$$
$$r := x1\, x2^2 + x2^3 - 2\, x1\, x2 + x1 \tag{4}$$

$src2 := SubresultantChain(r, t2, x2, R):$
$g2 := SubresultantOfIndex(1, src2, R);\ s := SubresultantOfIndex(0, src2, R);$

$$g2 := -2\, x1^3\, x2 + x1^3 - 5\, x1^2\, x2 - 5\, x1\, x2 - x1 - x2 + 2$$
$$s := x1^5 + 9\, x1^4 + 24\, x1^3 + 38\, x1^2 + 13\, x1 + 8 \tag{5}$$

$$sol := Chain([s], Empty(R), R) : IsRegular(Initial(g2, R), sol, R);$$
$$true \tag{6}$$
$$sol2 := Chain([g2], sol, R) : IsRegular(Initial(g3, R), sol2, R);$$
$$true \tag{7}$$
$$IsRegular(Initial(t3, R), sol2, R);$$
$$true \tag{8}$$

$sol := Chain([s], Empty(R), R) : IsRegular(Initial(g2, R), sol, R);$

$$true \tag{6}$$

$sol2 := Chain([g2], sol, R) : IsRegular(Initial(g3, R), sol2, R);$

$$true \tag{7}$$

$IsRegular(Initial(t3, R), sol2, R);$

$$true \tag{8}$$

$sol3 := Chain([g3], sol2, R) : Display(sol3, R);$

$$\begin{cases} (x2^2 + x2\,x1 - x1)\,x3 + x2 = 0 \\ (-2\,x1^3 - 5\,x1^2 - 5\,x1 - 1)\,x2 + x1^3 - x1 + 2 = 0 \\ x1^5 + 9\,x1^4 + 24\,x1^3 + 38\,x1^2 + 13\,x1 + 8 = 0 \\ x2^2 + x2\,x1 - x1 \neq 0 \\ -2\,x1^3 - 5\,x1^2 - 5\,x1 - 1 \neq 0 \end{cases} \tag{9}$$

$sol := Chain([s], Empty(R), R) : IsRegular(Initial(g2, R), sol, R);$

$$true \tag{6}$$

$sol2 := Chain([g2], sol, R) : IsRegular(Initial(g3, R), sol2, R);$

$$true \tag{7}$$

$IsRegular(Initial(t3, R), sol2, R);$

$$true \tag{8}$$

$sol3 := Chain([g3], sol2, R) : Display(sol3, R);$

$$\begin{cases} (x2^2 + x2\,x1 - x1)\,x3 + x2 = 0 \\ (-2\,x1^3 - 5\,x1^2 - 5\,x1 - 1)\,x2 + x1^3 - x1 + 2 = 0 \\ x1^5 + 9\,x1^4 + 24\,x1^3 + 38\,x1^2 + 13\,x1 + 8 = 0 \\ x2^2 + x2\,x1 - x1 \neq 0 \\ -2\,x1^3 - 5\,x1^2 - 5\,x1 - 1 \neq 0 \end{cases} \tag{9}$$

$dec3 := Triangularize([f, t3, t2], R) : Display(dec3[1], R);$

$$\begin{cases} (x2^2 + x2\,x1 - x1)\,x3 + x2 = 0 \\ (2\,x1^3 + 5\,x1^2 + 5\,x1 + 1)\,x2 - x1^3 + x1 - 2 = 0 \\ x1^5 + 9\,x1^4 + 24\,x1^3 + 38\,x1^2 + 13\,x1 + 8 = 0 \\ x2^2 + x2\,x1 - x1 \neq 0 \\ 2\,x1^3 + 5\,x1^2 + 5\,x1 + 1 \neq 0 \end{cases} \tag{10}$$

The modular algorithm in $\mathbb{F}_p$ under our hypotheses

1. Evaluate f and T at sufficiently many (use the Bézout bound or the mixed volume) values a of X_1 so that T specializes well at $X_1 = a$ to a zero-dimensional regular chain T_a

The modular algorithm in $\mathbb{F}_p$ under our hypotheses

1. Evaluate f and T at sufficiently many (use the Bézout bound or the mixed volume) values a of X_1 so that T specializes well at $X_1 = a$ to a zero-dimensional regular chain T_a
2. For each good specialization $X_1 = a$

The modular algorithm in $\mathbb{F}_p$ under our hypotheses

1. Evaluate f and T at sufficiently many (use the Bézout bound or the mixed volume) values a of X_1 so that T specializes well at $X_1 = a$ to a zero-dimensional regular chain T_a
2. For each good specialization $X_1 = a$
   1. Replace T_a by a normalized $(=$ monic) regular chain N_a

The modular algorithm in $\mathbb{F}_p$ under our hypotheses

1. Evaluate f and T at sufficiently many (use the Bézout bound or the mixed volume) values a of X_1 so that T specializes well at $X_1 = a$ to a zero-dimensional regular chain T_a
2. For each good specialization $X_1 = a$
   1. Replace T_a by a normalized (= monic) regular chain N_a
   2. Compute the images of the polynomials r_{i-1} and g_i at $X_1 = a$

The modular algorithm in $\mathbb{F}_p$ under our hypotheses

1. Evaluate f and T at sufficiently many (use the Bézout bound or the mixed volume) values a of X_1 so that T specializes well at $X_1 = a$ to a zero-dimensional regular chain T_a
2. For each good specialization $X_1 = a$
    1. Replace T_a by a normalized (= monic) regular chain N_a
    2. Compute the images of the polynomials r_{i-1} and g_i at $X_1 = a$
3. Recover X_1 (by interpolation and rational function reconstruction) and deduce $\overline{s}, g_2, \ldots, g_n$

The modular algorithm in $\mathbb{F}_p$ under our hypotheses

1. Evaluate f and T at sufficiently many (use the Bézout bound or the mixed volume) values a of X_1 so that T specializes well at $X_1 = a$ to a zero-dimensional regular chain T_a
2. For each good specialization $X_1 = a$
    1. Replace T_a by a normalized (= monic) regular chain N_a
    2. Compute the images of the polynomials r_{i-1} and g_i at $X_1 = a$
3. Recover X_1 (by interpolation and rational function reconstruction) and deduce $\overline{s}, g_2, \ldots, g_n$

Technical details:

- specializations $X_1 = a$, $X_1 = b$, ... must produce faithful images of resultants r_i, that is, resultants of maximum degree. good $\neq$ faithful.

The modular algorithm in $\mathbb{F}_p$ under our hypotheses

1. Evaluate f and T at sufficiently many (use the Bézout bound or the mixed volume) values a of X_1 so that T specializes well at $X_1 = a$ to a zero-dimensional regular chain T_a
2. For each good specialization $X_1 = a$
   1. Replace T_a by a normalized (= monic) regular chain N_a
   2. Compute the images of the polynomials r_{i-1} and g_i at $X_1 = a$
3. Recover X_1 (by interpolation and rational function reconstruction) and deduce $\overline{s}, g_2, \ldots, g_n$

Technical details:

- specializations $X_1 = a$, $X_1 = b$, ... must produce faithful images of resultants r_i, that is, resultants of maximum degree. good $\neq$ faithful.
- the implementation uses a priori bounds for
  1. the number of non-faithful specializations, and
  2. the degree of $\overline{s}$; see the details in our CASC 2023 paper.

The modular algorithm in $\mathbb{F}_p$ under our hypotheses

1. Evaluate f and T at sufficiently many (use the Bézout bound or the mixed volume) values a of X_1 so that T specializes well at $X_1 = a$ to a zero-dimensional regular chain T_a
2. For each good specialization $X_1 = a$
   1. Replace T_a by a normalized (= monic) regular chain N_a
   2. Compute the images of the polynomials r_{i-1} and g_i at $X_1 = a$
3. Recover X_1 (by interpolation and rational function reconstruction) and deduce $\overline{s}, g_2, \ldots, g_n$

Technical details:

- specializations $X_1 = a$, $X_1 = b$, ... must produce faithful images of resultants r_i, that is, resultants of maximum degree. good $\neq$ faithful.
- the implementation uses a priori bounds for
  1. the number of non-faithful specializations, and
  2. the degree of $\overline{s}$; see the details in our CASC 2023 paper.
- we stop combining those images of the r_i's when the recombination of the images stabilizes (Monagan's probabilistic idea, ISSAC 2005).

The full modular algorithm: relaxing the hypotheses (1/3)

H1 for $1 \leq i \leq n-1$, we have $r_i \notin \mathbf{k}$ and $\mathrm{mvar}(r_i) = X_i$,

H2 For $2 \leq i \leq n$, we have $g_i \notin \mathbf{k}$ and $\mathrm{mvar}(g_i) = X_i$,

H3 The polynomial set $C := \{\overline{s}, g_2, \ldots g_n\}$ is a regular chain,

H4 For every $2 \leq i \leq n$, $\mathrm{lc}(t_i, X_i)$ is invertible modulo $\langle \overline{s}, g_2, \ldots, g_{i-1} \rangle$.

The full modular algorithm: relaxing the hypotheses (1/3)

- H1 for $1 \le i \le n - 1$, we have $r_i \notin \mathbf{k}$ and $\mathrm{mvar}(r_i) = X_i$,
- H2 For $2 \le i \le n$, we have $g_i \notin \mathbf{k}$ and $\mathrm{mvar}(g_i) = X_i$,
- H3 The polynomial set $C := \{\overline{s}, g_2, \ldots g_n\}$ is a regular chain,
- H4 For every $2 \le i \le n$, $\mathrm{lc}(t_i, X_i)$ is invertible modulo $\langle \overline{s}, g_2, \ldots, g_{i-1} \rangle$.

- Relaxing the $r_i \notin \mathbf{k}$ part of H1 implies that the last computed resultant, say s, could be constant:

The full modular algorithm: relaxing the hypotheses (1/3)

H1 for $1 \leq i \leq n-1$, we have $r_i \notin \mathbf{k}$ and $\mathrm{mvar}(r_i) = X_i$,

H2 For $2 \leq i \leq n$, we have $g_i \notin \mathbf{k}$ and $\mathrm{mvar}(g_i) = X_i$,

H3 The polynomial set $C := \{\overline{s}, g_2, \ldots g_n\}$ is a regular chain,

H4 For every $2 \leq i \leq n$, $\mathrm{lc}(t_i, X_i)$ is invertible modulo $\langle \overline{s}, g_2, \ldots, g_{i-1} \rangle$.

- Relaxing the $r_i \notin \mathbf{k}$ part of H1 implies that the last computed resultant, say s, could be constant:
  1. if $s = 0$, then its "parents" (say r_j and t_j) have a non-trivial GCD over $\mathbf{k}$ which must be added to the chain C,

The full modular algorithm: relaxing the hypotheses (1/3)

H1 for $1 \le i \le n-1$, we have $r_i \notin \mathbf{k}$ and $\mathrm{mvar}(r_i) = X_i$,

H2 For $2 \le i \le n$, we have $g_i \notin \mathbf{k}$ and $\mathrm{mvar}(g_i) = X_i$,

H3 The polynomial set $C := \{\overline{s}, g_2, \ldots g_n\}$ is a regular chain,

H4 For every $2 \le i \le n$, $\mathrm{lc}(t_i, X_i)$ is invertible modulo $\langle \overline{s}, g_2, \ldots, g_{i-1} \rangle$.

- Relaxing the $r_i \notin \mathbf{k}$ part of H1 implies that the last computed resultant, say s, could be constant:
  1. if $s = 0$, then its "parents" (say r_j and t_j) have a non-trivial GCD over $\mathbf{k}$ which must be added to the chain C,
  2. if $s \neq 0$, then Intersect$(f, T) = \varnothing$.

The full modular algorithm: relaxing the hypotheses (1/3)

H1 for $1 \leq i \leq n-1$, we have $r_i \notin \mathbf{k}$ and $\mathrm{mvar}(r_i) = X_i$,

H2 For $2 \leq i \leq n$, we have $g_i \notin \mathbf{k}$ and $\mathrm{mvar}(g_i) = X_i$,

H3 The polynomial set $C := \{\overline{s}, g_2, \ldots g_n\}$ is a regular chain,

H4 For every $2 \leq i \leq n$, $\mathrm{lc}(t_i, X_i)$ is invertible modulo $\langle \overline{s}, g_2, \ldots, g_{i-1} \rangle$.

- Relaxing the $r_i \notin \mathbf{k}$ part of H1 implies that the last computed resultant, say s, could be constant:
  1. if $s = 0$, then its "parents" (say r_j and t_j) have a non-trivial GCD over $\mathbf{k}$ which must be added to the chain C,
  2. if $s \neq 0$, then Intersect$(f, T) = \varnothing$.
- Handling this modification only requires to possibly computing this GCD, whose cost is negligible.

The full modular algorithm: relaxing the hypotheses (2/3)

H1 for $1 \leq i \leq n-1$, we have $r_i \notin \mathbf{k}$ and $\mathrm{mvar}(r_i) = X_i$,

H2 For $2 \leq i \leq n$, we have $g_i \notin \mathbf{k}$ and $\mathrm{mvar}(g_i) = X_i$,

H3 The polynomial set $C := \{\overline{s}, g_2, \ldots g_n\}$ is a regular chain,

H4 For every $2 \leq i \leq n$, $\mathrm{lc}(t_i, X_i)$ is invertible modulo $\langle \overline{s}, g_2, \ldots, g_{i-1} \rangle$.

The full modular algorithm: relaxing the hypotheses (2/3)

H1 for $1 \leq i \leq n-1$, we have $r_i \notin \mathbf{k}$ and $\mathrm{mvar}(r_i) = X_i$,

H2 For $2 \leq i \leq n$, we have $g_i \notin \mathbf{k}$ and $\mathrm{mvar}(g_i) = X_i$,

H3 The polynomial set $C := \{\overline{s}, g_2, \ldots g_n\}$ is a regular chain,

H4 For every $2 \leq i \leq n$, $\mathrm{lc}(t_i, X_i)$ is invertible modulo $\langle \overline{s}, g_2, \ldots, g_{i-1} \rangle$.

- Relaxing the $\mathrm{mvar}(r_i) = X_i$ part of H1 implies that the successive resultants $r_n, \ldots$ may have a gap in their sequence of main variables

The full modular algorithm: relaxing the hypotheses (2/3)

H1 for $1 \leq i \leq n-1$, we have $r_i \notin \mathbf{k}$ and $\mathrm{mvar}(r_i) = X_i$,

H2 For $2 \leq i \leq n$, we have $g_i \notin \mathbf{k}$ and $\mathrm{mvar}(g_i) = X_i$,

H3 The polynomial set $C := \{\overline{s}, g_2, \ldots g_n\}$ is a regular chain,

H4 For every $2 \leq i \leq n$, $\mathrm{lc}(t_i, X_i)$ is invertible modulo $\langle \overline{s}, g_2, \ldots, g_{i-1} \rangle$.

- Relaxing the $\mathrm{mvar}(r_i) = X_i$ part of H1 implies that the successive resultants $r_n, \ldots$ may have a gap in their sequence of main variables
- we simply use the appropriate polynomials from $T = \{t_n, \ldots, t_2\}$ to fill those gaps.

The full modular algorithm: relaxing the hypotheses (2/3)

H1 for $1 \leq i \leq n-1$, we have $r_i \notin \mathbf{k}$ and $\mathrm{mvar}(r_i) = X_i$,

H2 For $2 \leq i \leq n$, we have $g_i \notin \mathbf{k}$ and $\mathrm{mvar}(g_i) = X_i$,

H3 The polynomial set $C := \{\overline{s}, g_2, \ldots g_n\}$ is a regular chain,

H4 For every $2 \leq i \leq n$, $\mathrm{lc}(t_i, X_i)$ is invertible modulo $\langle \overline{s}, g_2, \ldots, g_{i-1} \rangle$.

- Relaxing the $\mathrm{mvar}(r_i) = X_i$ part of H1 implies that the successive resultants $r_n, \ldots$ may have a gap in their sequence of main variables
- we simply use the appropriate polynomials from $T = \{t_n, \ldots, t_2\}$ to fill those gaps.
- Handling this modification comes at no cost.

The full modular algorithm: relaxing the hypotheses (3/3)

H1 for $1 \le i \le n-1$, we have $r_i \notin \mathbf{k}$ and $\mathrm{mvar}(r_i) = X_i$,

H2 For $2 \le i \le n$, we have $g_i \notin \mathbf{k}$ and $\mathrm{mvar}(g_i) = X_i$,

H3 The polynomial set $C := \{\overline{s}, g_2, \ldots g_n\}$ is a regular chain,

H4 For every $2 \le i \le n$, $\mathrm{lc}(t_i, X_i)$ is invertible modulo $\langle \overline{s}, g_2, \ldots, g_{i-1} \rangle$.

The full modular algorithm: relaxing the hypotheses (3/3)

H1 for $1 \le i \le n-1$, we have $r_i \notin \mathbf{k}$ and $\mathrm{mvar}(r_i) = X_i$,

H2 For $2 \le i \le n$, we have $g_i \notin \mathbf{k}$ and $\mathrm{mvar}(g_i) = X_i$,

H3 The polynomial set $C := \{\overline{s}, g_2, \ldots g_n\}$ is a regular chain,

H4 For every $2 \le i \le n$, $\mathrm{lc}(t_i, X_i)$ is invertible modulo $\langle \overline{s}, g_2, \ldots, g_{i-1} \rangle$.

- When either H2, H3, or H4 fails

The full modular algorithm: relaxing the hypotheses (3/3)

H1 for $1 \le i \le n-1$, we have $r_i \notin \mathbf{k}$ and $\mathrm{mvar}(r_i) = X_i$,

H2 For $2 \le i \le n$, we have $g_i \notin \mathbf{k}$ and $\mathrm{mvar}(g_i) = X_i$,

H3 The polynomial set $C := \{\overline{s}, g_2, \ldots g_n\}$ is a regular chain,

H4 For every $2 \le i \le n$, $\mathrm{lc}(t_i, X_i)$ is invertible modulo $\langle \overline{s}, g_2, \ldots, g_{i-1} \rangle$.

- When either H2, H3, or H4 fails
  1. the "candidate" regular chain C must split, and

The full modular algorithm: relaxing the hypotheses (3/3)

H1 for $1 \leq i \leq n-1$, we have $r_i \notin \mathbf{k}$ and $\mathrm{mvar}(r_i) = X_i$,

H2 For $2 \leq i \leq n$, we have $g_i \notin \mathbf{k}$ and $\mathrm{mvar}(g_i) = X_i$,

H3 The polynomial set $C := \{\overline{s}, g_2, \ldots g_n\}$ is a regular chain,

H4 For every $2 \leq i \leq n$, $\mathrm{lc}(t_i, X_i)$ is invertible modulo $\langle \overline{s}, g_2, \ldots, g_{i-1} \rangle$.

- When either H2, H3, or H4 fails
  1. the "candidate" regular chain C must split, and
  2. some subresultants of index higher than 1 must be used.

The full modular algorithm: relaxing the hypotheses (3/3)

H1 for $1 \leq i \leq n-1$, we have $r_i \notin \mathbf{k}$ and $\mathrm{mvar}(r_i) = X_i$,

H2 For $2 \leq i \leq n$, we have $g_i \notin \mathbf{k}$ and $\mathrm{mvar}(g_i) = X_i$,

H3 The polynomial set $C := \{\overline{s}, g_2, \ldots g_n\}$ is a regular chain,

H4 For every $2 \leq i \leq n$, $\mathrm{lc}(t_i, X_i)$ is invertible modulo $\langle \overline{s}, g_2, \ldots, g_{i-1} \rangle$.

- When either H2, H3, or H4 fails
  1. the "candidate" regular chain C must split, and
  2. some subresultants of index higher than 1 must be used.
- Costs for handling this:

The full modular algorithm: relaxing the hypotheses (3/3)

H1 for $1 \le i \le n-1$, we have $r_i \notin \mathbf{k}$ and $\mathrm{mvar}(r_i) = X_i$,

H2 For $2 \le i \le n$, we have $g_i \notin \mathbf{k}$ and $\mathrm{mvar}(g_i) = X_i$,

H3 The polynomial set $C := \{\overline{s}, g_2, \ldots g_n\}$ is a regular chain,

H4 For every $2 \le i \le n$, $\mathrm{lc}(t_i, X_i)$ is invertible modulo $\langle \overline{s}, g_2, \ldots, g_{i-1} \rangle$.

- When either H2, H3, or H4 fails
  1. the "candidate" regular chain C must split, and
  2. some subresultants of index higher than 1 must be used.
- Costs for handling this:
  1. computing resultants and GCDs modulo regular chains by evaluation and interpolation, which is what this whole algorithm is about,

The full modular algorithm: relaxing the hypotheses (3/3)

H1 for $1 \leq i \leq n-1$, we have $r_i \notin \mathbf{k}$ and $\mathrm{mvar}(r_i) = X_i$,

H2 For $2 \leq i \leq n$, we have $g_i \notin \mathbf{k}$ and $\mathrm{mvar}(g_i) = X_i$,

H3 The polynomial set $C := \{\overline{s}, g_2, \ldots g_n\}$ is a regular chain,

H4 For every $2 \leq i \leq n$, $\mathrm{lc}(t_i, X_i)$ is invertible modulo $\langle \overline{s}, g_2, \ldots, g_{i-1} \rangle$.

- When either H2, H3, or H4 fails
  1. the "candidate" regular chain C must split, and
  2. some subresultants of index higher than 1 must be used.
- Costs for handling this:
  1. computing resultants and GCDs modulo regular chains by evaluation and interpolation, which is what this whole algorithm is about,

The full modular algorithm: relaxing the hypotheses (3/3)

H1 for $1 \leq i \leq n-1$, we have $r_i \notin \mathbf{k}$ and $\mathrm{mvar}(r_i) = X_i$,

H2 For $2 \leq i \leq n$, we have $g_i \notin \mathbf{k}$ and $\mathrm{mvar}(g_i) = X_i$,

H3 The polynomial set $C := \{\overline{s}, g_2, \ldots g_n\}$ is a regular chain,

H4 For every $2 \leq i \leq n$, $\mathrm{lc}(t_i, X_i)$ is invertible modulo $\langle \overline{s}, g_2, \ldots, g_{i-1} \rangle$.

- When either H2, H3, or H4 fails
  1. the "candidate" regular chain C must split, and
  2. some subresultants of index higher than 1 must be used.
- Costs for handling this:
  1. computing resultants and GCDs modulo regular chains by evaluation and interpolation, which is what this whole algorithm is about,
  2. interpolating those subresultants of higher index

Outline

1. Triangular decompositions in polynomial system solving
2. Modular methods in polynomial system solving
3. A Modular methods for incremental triangular decompositions
4. Conclusions

Conclusions

- We have discussed Intersect(f, T) which computes $V(f) \cap W(T)$ and which is at the core of the incremental method for triangular decompositions
- We have presented a modular method for Intersect(f, T) focusing on the case where T is dimension one.
- This method allows us to get rid off of the large extraneous factors occurring in iterated resultant computations
- For technical details (in particular degree bounds) see our CASC 2023.
- The experimentation reported there is based on an implementation which does not support yet the relaxation of our hypotheses (thus providing no benefits when those hypotheses do not hold).

- This modular method is designed to take advantage of FFT-based algorithms (speculative methods for computing subresultant chains, see our CASC 2022 paper).
- Parallel execution: multiple specialization can be done concurrently.

Thank You!

Basic Polynomial Algebra Subprograms

http://www.bpaslib.org/

References

[1] M. Asadi, A. Brandt, R. H. C. Moir, and M. Moreno Maza. "Algorithms and Data Structures for Sparse Polynomial Arithmetic". In: *Mathematics* 7.5 (2019), p. 441.

[2] M. Asadi, A. Brandt, R. H. C. Moir, M. Moreno Maza, and Y. Xie. "On the parallelization of triangular decompositions". In: *International Symposium on Symbolic and Algebraic Computation (ISSAC '20), Kalamata, Greece, July 20-23, 2020*. ACM, 2020, pp. 22–29.

[3] M. Asadi, A. Brandt, R. H. C. Moir, M. Moreno Maza, and Y. Xie. "Parallelization of Triangular Decompositions: Techniques and Implementation". In: *J. Symb. Comput.* (2021). *(to appear)*.

[4] P. Aubry, D. Lazard, and M. Moreno Maza. "On the Theories of Triangular Sets". In: *J. Symb. Comput.* 28.1-2 (1999), pp. 105–124.

[5] F. Boulier, F. Lemaire, and M. Moreno Maza. "Well known theorems on triangular systems and the D5 principle". In: Transgressive Computing 2006*, Proc.* 2006.

[6] A. Brandt, R. H. C. Moir, and M. Moreno Maza. "Employing C++ Templates in the Design of a Computer Algebra Library". In: *Mathematical Software - ICMS 2020, Braunschweig, Germany, July 13-16, 2020*. Vol. 12097. LNCS. Springer, 2020, pp. 342–352.

[7] M. Moreno Maza and Y. Xie. "Balanced Dense Polynomial Multiplication on Multi-Cores". In: *Int. J. Found. Comput. Sci.* 22.5 (2011), pp. 1035–1055.

[8] J. F. Ritt. *Differential Algebra*. New York: Dover Publications, Inc., 1966.

[9] J. F. Ritt. *Differential Equations from an Algebraic Standpoint*. Vol. 14. New York: American Mathematical Society, 1932.

[10] W. T. Wu. "A Zero Structure Theorem for polynomial equations solving". In: *MM Research Preprints* 1 (1987), pp. 2–12.