Resultant

&

Newton iteration

Alin Bostan

MPRI C-2-22

October 11, 2022

The exercise from last week

Let f and g in $\mathbb{K}[x,y]$ have degrees at most d_x in x and at most d_y in y.

(a) Show that it is possible to compute the product $h = fg$ using

$$O(\mathsf{M}(d_x d_y))$$

arithmetic operations in $\mathbb{K}$.
Hint: Use the substitution $x \leftarrow y^{2d_y+1}$ to reduce the problem to the product of univariate polynomials.

(b) Improve this result by proposing an evaluation-interpolation scheme which allows the computation of h in

$$O(d_x \, \mathsf{M}(d_y) + d_y \, \mathsf{M}(d_x))$$

arithmetic operations in $\mathbb{K}$.

Solution of (a)

(a) Show that it is possible to compute $h = fg$ using $O(\mathsf{M}(d_x d_y))$ ops. in $\mathbb{K}$. *Hint*: Use the substitution $x \leftarrow y^{2d_y+1}$ to reduce the problem to the product of univariate polynomials.

Solution:

▷ Write $h(x, y) = h_0(y) + xh_1(y) + \cdots + x^{2d_x} h_{2d_x}(y)$ with $\deg_y h_i \leq 2d_y$ for $0 \leq i \leq 2d_x$ and observe that in the specialization $h(y^{2d_y+1}, y)$, the terms $y^{(2d_y+1)i} h_i(y)$ have distinct supports.

▷ So one gets $h(x, y)$ from $h(y^{2d_y+1}, y)$ in no arithmetic operation.

▷ Similarly, $f(y^{2d_y+1}, y)$ is obtained from $f(x, y)$ with no calculation, the same holds for g.

▷ The only needed calculation is $h(y^{2d_y+1}, y) = f(y^{2d_y+1}, y) \times g(y^{2d_y+1}, y)$, which requires $O(\mathsf{M}(d_x d_y))$ ops. in $\mathbb{K}$. □

Solution of (b)

(b) Improve this result by proposing an evaluation-interpolation scheme which allows the computation of h in $O(d_x\,\mathsf{M}(d_y) + d_y\,\mathsf{M}(d_x))$ ops. in $\mathbb{K}$.

Solution:

▷ Each polynomial $h_i(y)$ has degree $\leq 2d_y$ and so can be obtained by interpolation from values at $2d_y + 1$ points.

▷ To minimize costs, use $(1, q, q^2, \dots, q^{2d_y})$ and get evaluations of all $h_i(y)$ simultaneously. So first write $f(x, y) = f_0(y) + xf_1(y) + \cdots + x^{2d_x} f_{2d_x}(y)$ with $\deg_y f_i \leq d_y$ for $0 \leq i \leq d_x$ and similarly for $g(x, y)$.

- For $0 \leq i \leq d_x$, evaluate $f_i(y)$ and $g_i(y)$ at $(q^j)_{0 \leq j \leq 2d_y}$. $O(d_x\mathsf{M}(d_y))$
- For $0 \leq j \leq 2d_y$, do:
  - compute $f(x, q^j) = \sum_{i=0}^{d_x} x^i f_i(q^j)$;
  - compute $g(x, q^j) = \sum_{i=0}^{d_x} x^i g_i(q^j)$;
  - compute $h(x, q^j) = f(x, q^j) \times g(x, q^j)$. $O(d_y\mathsf{M}(d_x))$
- For $0 \leq i \leq 2d_x$, interpolate $(h_i(q^j))_{0 \leq j \leq 2d_y}$ to get $h_i(y)$. $O(d_x\mathsf{M}(d_y))$
- Return $h(x, y) = \sum_{i=0}^{2d_x} x^i h_i(y)$.

Resultant

Definition

The Sylvester matrix of $A = a_m x^m + \cdots + a_0 \in \mathbb{K}[x]$, $(a_m \neq 0)$, and of $B = b_n x^n + \cdots + b_0 \in \mathbb{K}[x]$, $(b_n \neq 0)$, is the square matrix of size $m+n$

$$\mathsf{Syl}(A,B) = \begin{bmatrix} a_m & a_{m-1} & \dots & a_0 & & & \\ & a_m & a_{m-1} & \dots & a_0 & & \\ & & \ddots & \ddots & & \ddots & \\ & & & a_m & a_{m-1} & \dots & a_0 \\ b_n & b_{n-1} & \dots & b_0 & & & \\ & b_n & b_{n-1} & \dots & b_0 & & \\ & & \ddots & \ddots & & \ddots & \\ & & & b_n & b_{n-1} & \dots & b_0 \end{bmatrix}$$

The resultant $\mathsf{Res}(A,B)$ of A and B is the determinant of $\mathsf{Syl}(A,B)$.

▷ Definition extends to polynomials over any commutative ring R.

Key observation

If $\quad A = a_m x^m + \cdots + a_0 \quad$ and $\quad B = b_n x^n + \cdots + b_0, \quad$ then

$$\begin{bmatrix} a_m & a_{m-1} & \dots & a_0 & & \\ & \ddots & \ddots & & \ddots & \\ & & a_m & a_{m-1} & \dots & a_0 \\ b_n & b_{n-1} & \dots & b_0 & & \\ & \ddots & \ddots & & \ddots & \\ & & b_n & b_{n-1} & \dots & b_0 \end{bmatrix} \times \begin{bmatrix} \alpha^{m+n-1} \\ \vdots \\ \alpha \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha^{n-1}A(\alpha) \\ \vdots \\ A(\alpha) \\ \alpha^{m-1}B(\alpha) \\ \vdots \\ B(\alpha) \end{bmatrix}$$

Corollary: If $A(\alpha) = B(\alpha) = 0$, then $\mathsf{Res}\,(A, B) = 0$.

Example: the discriminant

The discriminant of A is the resultant of A and of its derivative A'.

E.g. for $A = ax^2 + bx + c$,

$$\mathsf{Disc}(A) = \mathsf{Res}\,(A, A') = \det \begin{bmatrix} a & b & c \\ 2a & b & \\ & 2a & b \end{bmatrix} = -a(b^2 - 4ac).$$

E.g. for $A = ax^3 + bx + c$,

$$\mathsf{Disc}(A) = \mathsf{Res}\,(A, A') = \det \begin{bmatrix} a & 0 & b & c & \\ & a & 0 & b & c \\ 3a & 0 & b & & \\ & 3a & 0 & b & \\ & & 3a & 0 & b \end{bmatrix} = a^2(4b^3 + 27ac^2).$$

▷ The discriminant vanishes when A and A' have a common root, that is when A has a multiple root.

Main properties

- Link with gcd Res $(A, B) = 0$ if and only if $\gcd(A, B)$ is non-constant.

- Elimination property

  There exist $U, V \in \mathbb{K}[x]$ not both zero, with $\deg(U) < n$, $\deg(V) < m$ and such that the following Bézout identity holds in $\mathbb{K} \cap (A, B)$:

$$\text{Res}\,(A, B) = UA + VB.$$

- Poisson formula

  If $A = a(x - \alpha_1) \cdots (x - \alpha_m)$ and $B = b(x - \beta_1) \cdots (x - \beta_n)$, then

$$\text{Res}\,(A, B) \;=\; a^n b^m \prod_{i,j} (\alpha_i - \beta_j) \;=\; a^n \prod_{1 \le i \le m} B(\alpha_i).$$

- Multiplicativity

$$\text{Res}\,(A \cdot B, C) = \text{Res}\,(A, C) \cdot \text{Res}\,(B, C), \quad \text{Res}\,(A, B \cdot C) = \text{Res}\,(A, B) \cdot \text{Res}\,(A, C).$$

Proof of Poisson's formula

▷ Direct consequence of the key observation:

If $A = (x-\alpha_1)\cdots(x-\alpha_m)$ and $B = (x-\beta_1)\cdots(x-\beta_n)$ then

$$\mathsf{Syl}(A,B) \times \begin{bmatrix} \beta_1^{m+n-1} & \dots & \beta_n^{m+n-1} & \alpha_1^{m+n-1} & \dots & \alpha_m^{m+n-1} \\ \vdots & & \vdots & \vdots & & \vdots \\ \beta_1 & \dots & \beta_n & \alpha_1 & \dots & \alpha_m \\ 1 & \dots & 1 & 1 & \dots & 1 \end{bmatrix} =$$

$$= \begin{bmatrix} \beta_1^{n-1}A(\beta_1) & \dots & \beta_n^{n-1}A(\beta_n) & 0 & \dots & 0 \\ \vdots & & \vdots & \vdots & & \vdots \\ A(\beta_1) & \dots & A(\beta_n) & 0 & \dots & 0 \\ 0 & \dots & 0 & \alpha_1^{m-1}B(\alpha_1) & \dots & \alpha_m^{m-1}B(\alpha_m) \\ \vdots & & \vdots & \vdots & & \vdots \\ 0 & \dots & 0 & B(\alpha_1) & \dots & B(\alpha_m) \end{bmatrix}$$

▷ To conclude, take determinants and use Vandermonde's formula

Application: computation with algebraic numbers

Let $A = \prod_i (x - \alpha_i)$ and $B = \prod_j (x - \beta_j)$ be polynomials of $\mathbb{K}[x]$. Then

$$A \oplus B := \prod_{i,j} (t - (\alpha_i + \beta_j)) = \mathsf{Res}_x(A(x), B(t-x)),$$

$$\prod_{i,j} (t - (\beta_j - \alpha_i)) = \mathsf{Res}_x(A(x), B(t+x)),$$

$$A \otimes B := \prod_{i,j} (t - \alpha_i \beta_j) = \mathsf{Res}_x(A(x), x^{\deg B} B(t/x)),$$

$$\prod_{i} (t - B(\alpha_i)) = \mathsf{Res}_x(A(x), t - B(x)).$$

In particular, the set $\overline{\mathbb{Q}}$ of algebraic numbers is a field.

Proof: Poisson's formula. E.g., first one: $\prod_i B(t - \alpha_i) = \prod_{i,j} (t - \alpha_i - \beta_j)$.

A beautiful identity of Ramanujan's

$$\frac{\sin\frac{2\pi}{7}}{\sin^2\frac{3\pi}{7}} - \frac{\sin\frac{\pi}{7}}{\sin^2\frac{2\pi}{7}} + \frac{\sin\frac{3\pi}{7}}{\sin^2\frac{\pi}{7}} = 2\sqrt{7}.$$

- ▷ If $p = \pi/7$ then $\sin(kp) = (\alpha^k - \alpha^{-k})/(2\,i)$, where $\alpha = e^{ip}$, with $\alpha^7 = -1$
- ▷ Since $\alpha \in \overline{\mathbb{Q}}$, any rational expression in the $\sin(kp)$ is in $\mathbb{Q}(i)(\alpha)$ thus in $\overline{\mathbb{Q}}$

```
> f:=sin(2*p)/sin(3*p)^2-sin(p)/sin(2*p)^2+sin(3*p)/sin(p)^2:
> expand(convert(f, exp)):
> F:=normal(subs(exp(I*p)=alpha, %));
```

$$\frac{2\,i\,(\alpha^{16} + 5\,\alpha^{14} + 12\,\alpha^{12} + \alpha^{11} + 20\,\alpha^{10} + 3\,\alpha^9 + 23\,\alpha^8 + 3\,\alpha^7 + 20\,\alpha^6 + \alpha^5 + 12\,\alpha^4 + 5\,\alpha^2 + 1)}{\alpha\,(\alpha^2 - 1)\,(\alpha^2 + 1)^2\,(\alpha^4 + \alpha^2 + 1)^2}$$

- ▷ In particular our LHS, $F(\alpha) = \frac{N(\alpha)}{D(\alpha)}$, is an algebraic number
- ▷ Resultant $R(t) := \mathrm{Res}_x(x^7 + 1, t \cdot D(x) - N(x))$ annihilates $F(\alpha)$

```
> R:=factor(resultant(x^7+1, t*denom(F)-numer(F), x));
```

$$-1274\,i\,\left(t^2 - 28\right)^3$$

Shanks' 1974 identities

$$\sqrt{11+2\sqrt{29}}+\sqrt{16-2\sqrt{29}+2\sqrt{55-10\sqrt{29}}}=\sqrt{5}+\sqrt{22+2\sqrt{5}}$$

$$\sqrt{\sqrt{m+n}+\sqrt{n}}+\sqrt{\sqrt{m+n}+m-\sqrt{n}+2\sqrt{m\left(\sqrt{m+n}-\sqrt{n}\right)}}$$

$$=\sqrt{m}+\sqrt{2\sqrt{m+n}+2\sqrt{m}}$$

A first exercise for next Tuesday

(1) The aim of this exercise is to prove algorithmically the following identity:

$$\sqrt[3]{\sqrt[3]{2}-1} = \sqrt[3]{\frac{1}{9}} - \sqrt[3]{\frac{2}{9}} + \sqrt[3]{\frac{4}{9}}. \qquad \text{(E)}$$

Let $a = \sqrt[3]{2}$ and $b = \sqrt[3]{\frac{1}{9}}$.

(a) Determine $P_c \in \mathbb{Q}[x]$ annihilating $c = 1 - a + a^2$, using a resultant.

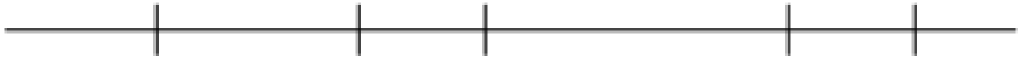
(b) Deduce $P_R \in \mathbb{Q}[x]$ annihilating the RHS of (E), by another resultant.

(c) Show that the polynomial computed in (b) also annihilates the LHS of (E).

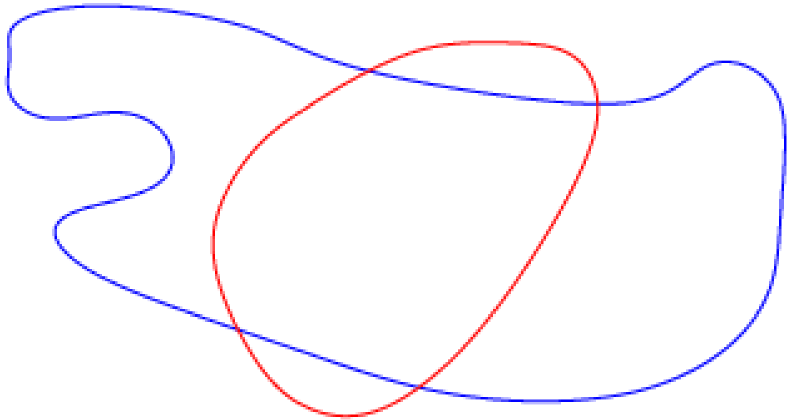
(d) Conclude.

Systems of two equations and two unknowns

Geometrically, roots of a polynomial $f \in \mathbb{Q}[x]$ correspond to points on a line.



Roots of polynomials $A \in \mathbb{Q}[x, y]$ correspond to plane curves $A = 0$.

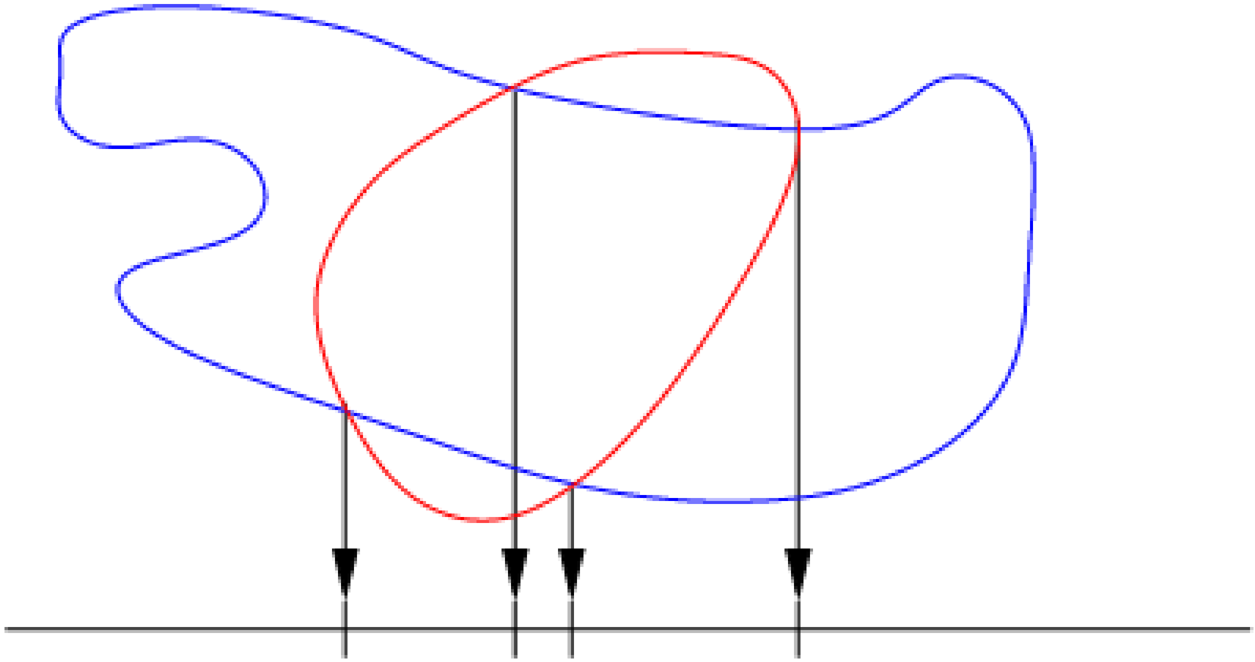


Let now A and B be in $\mathbb{Q}[x, y]$. Then:

- either the curves $A = 0$ and $B = 0$ have a common component,
- or they intersect in a finite number of points.

Application: Resultants compute projections

Theorem. Let $A = a_m y^m + \cdots$ and $B = b_n y^n + \cdots$ be polynomials in $\mathbb{Q}[x][y]$. The roots of $\mathsf{Res}_y(A, B) \in \mathbb{Q}[x]$ are either the abscissas of points in the intersection $A = B = 0$, or common roots of a_m and b_n.



Proof. Elimination property: $\mathsf{Res}\,(A, B) = UA + VB$, for $U, V \in \mathbb{Q}[x, y]$.

Thus $A(\alpha, \beta) = B(\alpha, \beta) = 0$ implies $\mathsf{Res}_y(A, B)(\alpha) = 0$

Application: implicitization of parametric curves

Task: Given a rational parametrization of a curve

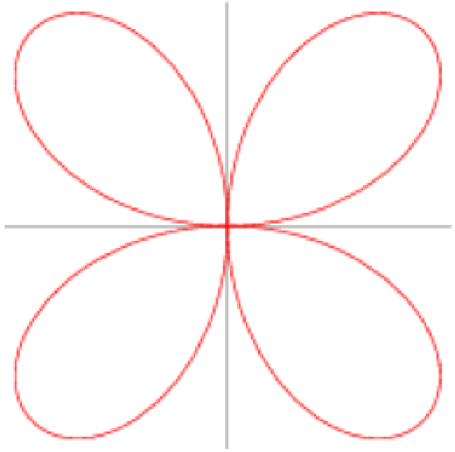
$$x = A(t), \quad y = B(t), \qquad A, B \in \mathbb{K}(t),$$

compute a non-trivial polynomial in x and y vanishing on the curve.

Recipe: take the resultant in t of numerators of $x - A(t)$ and $y - B(t)$.

Example: for the four-leaved clover (a.k.a. quadrifolium) given by

$$x = \frac{4t(1-t^2)^2}{(1+t^2)^3}, \quad y = \frac{8t^2(1-t^2)}{(1+t^2)^3},$$



$$\mathsf{Res}_t((1+t^2)^3x-4t(1-t^2)^2,(1+t^2)^3y-8t^2(1-t^2)) = 2^{24}\left((x^2+y^2)^3-4x^2y^2\right).$$

Computation of the resultant

An Euclidean-type algorithm for the resultant bases on:

- If $A = QB + R$, and $R \neq 0$, then (by Poisson's formula)

$$\mathsf{Res}\,(A, B) = (-1)^{\deg A \deg B}\, \mathsf{lc}(B)^{\deg A - \deg R}\, \mathsf{Res}\,(B, R).$$

- If B is constant, then $\mathsf{Res}\,(A, B) = B^{\deg A}$.

If $(R_0, \ldots, R_{N-1}, R_N = \gcd(A, B), 0)$ is the remainder sequence produced by the Euclidean algorithm for $R_0 = A$ and $R_1 = B$, then

- either $\deg R_N$ is non-constant, and $\mathsf{Res}\,(A, B) = 0$,
- or $\mathsf{Res}\,(A, B) = R_N^{\deg R_{N-1}} \prod_{i=0}^{N-2} (-1)^{\deg R_i \deg R_{i+1}}\, \mathsf{lc}(R_{i+1})^{\deg R_i - \deg R_{i+2}}$.

▷ This leads to a $O(N^2)$ algorithm for $\mathsf{Res}\,(A, B)$, where $\deg(A), \deg(B) \leq N$.

▷ A divide-and-conquer $O(\mathsf{M}(N) \log N)$ algorithm requires extra-work.

Bonus

1. Fast Manipulation of Algebraic Numbers

ELSEVIER

Available online at www.sciencedirect.com

SCIENCE DIRECT®

Journal of Symbolic Computation 41 (2006) 1–29

Journal of Symbolic Computation

www.elsevier.com/locate/jsc

Fast computation of special resultants

Alin Bostan[a,*], Philippe Flajolet[a], Bruno Salvy[a], Éric Schost[b]

[a] *Algorithms Project, Inria Rocquencourt, 78153 Le Chesnay, France*
[b] *LIX, École polytechnique, 91128 Palaiseau, France*

Received 3 September 2003; accepted 9 July 2005
Available online 25 October 2005

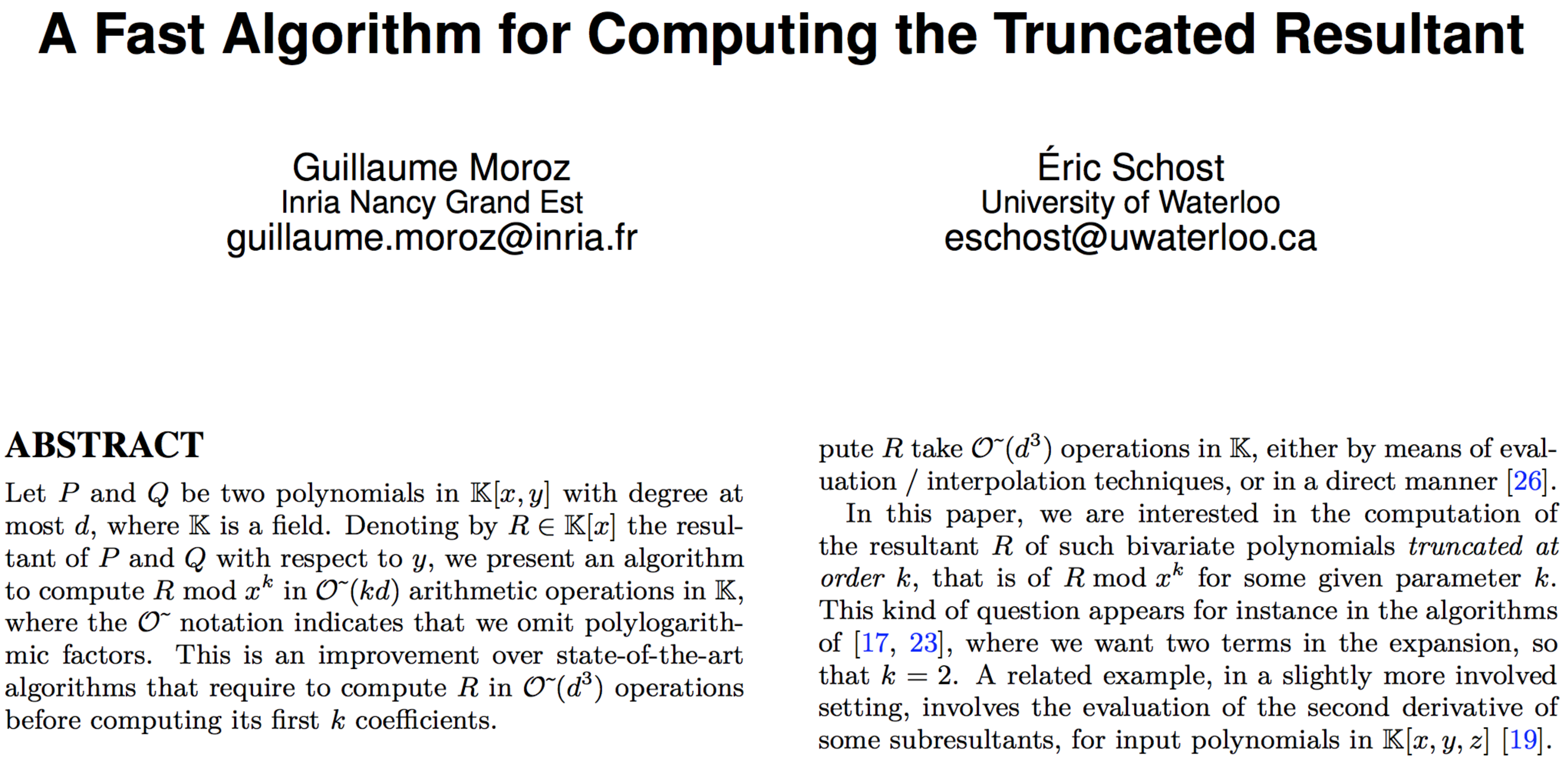
Abstract

We propose fast algorithms for computing *composed products* and *composed sums*, as well as *diamond products* of univariate polynomials. These operations correspond to special multivariate resultants, that we compute using power sums of roots of polynomials, by means of their generating series.

Keywords: Diamond product; Composed product; Composed sum; Complexity; Tellegen's principle

▷ Composed sum $A \oplus B$ and composed product $A \otimes B$ in $\tilde{O}(\deg A \cdot \deg B)$

2. Computing the Truncated Resultant



A Fast Algorithm for Computing the Truncated Resultant

Guillaume Moroz
Inria Nancy Grand Est
guillaume.moroz@inria.fr

Éric Schost
University of Waterloo
eschost@uwaterloo.ca

ABSTRACT

Let P and Q be two polynomials in $\mathbb{K}[x,y]$ with degree at most d, where $\mathbb{K}$ is a field. Denoting by $R \in \mathbb{K}[x]$ the resultant of P and Q with respect to y, we present an algorithm to compute $R \bmod x^k$ in $\tilde{O}(kd)$ arithmetic operations in $\mathbb{K}$, where the $\tilde{O}$ notation indicates that we omit polylogarithmic factors. This is an improvement over state-of-the-art algorithms that require to compute R in $\tilde{O}(d^3)$ operations before computing its first k coefficients.

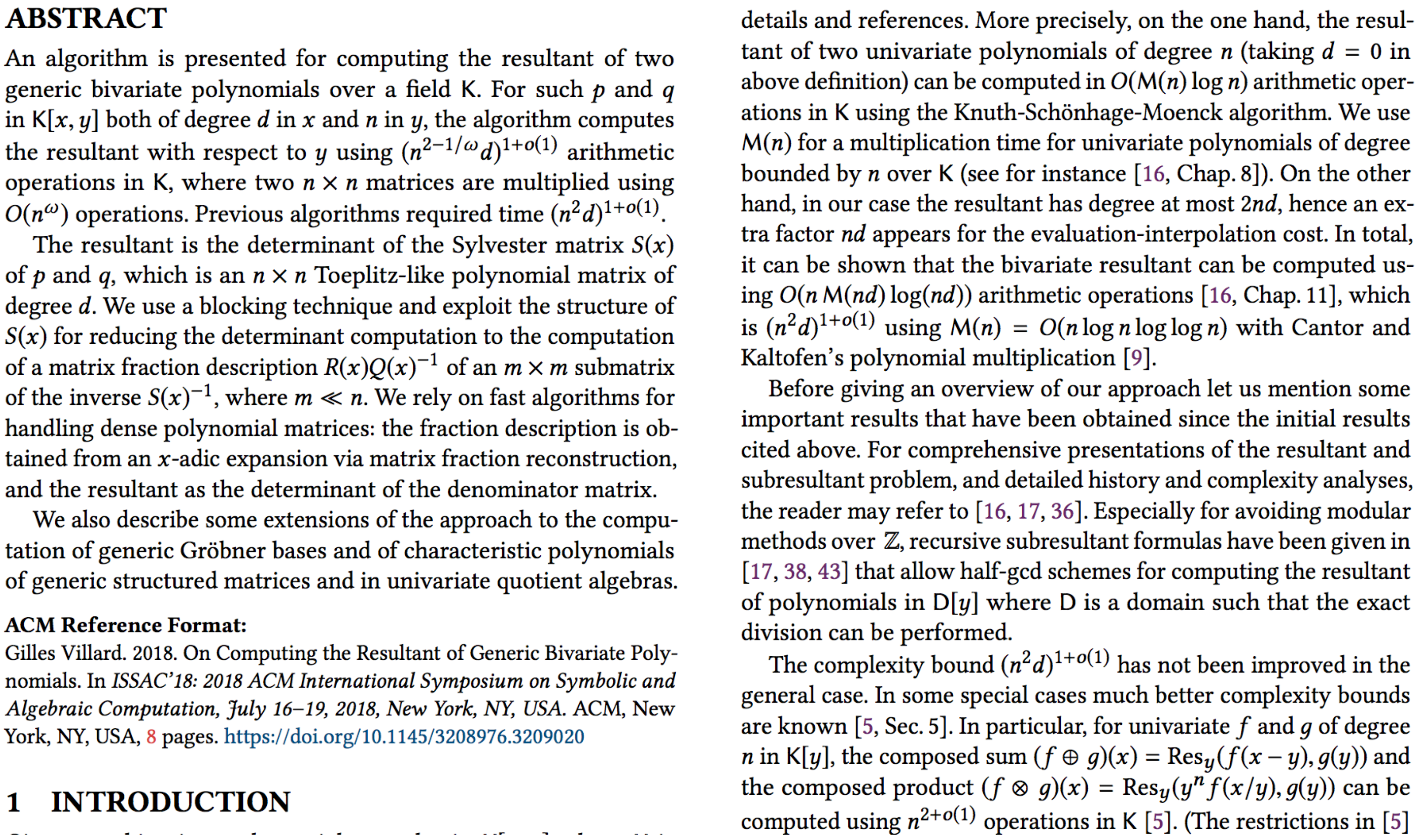
pute R take $\tilde{O}(d^3)$ operations in $\mathbb{K}$, either by means of evaluation / interpolation techniques, or in a direct manner [26].

In this paper, we are interested in the computation of the resultant R of such bivariate polynomials *truncated at order* k, that is of $R \bmod x^k$ for some given parameter k. This kind of question appears for instance in the algorithms of [17, 23], where we want two terms in the expansion, so that $k = 2$. A related example, in a slightly more involved setting, involves the evaluation of the second derivative of some subresultants, for input polynomials in $\mathbb{K}[x,y,z]$ [19].

[Moroz & Schost, ISSAC 2016]

▷ $\mathrm{Res}_y(P(x,y), Q(x,y)) \bmod x^k$ in $\tilde{O}(kd)$, where $d = \max(\deg P, \deg Q)$

3. Resultant of Generic Bivariate Polynomials



ABSTRACT

An algorithm is presented for computing the resultant of two generic bivariate polynomials over a field K. For such p and q in $K[x, y]$ both of degree d in x and n in y, the algorithm computes the resultant with respect to y using $(n^{2-1/\omega}d)^{1+o(1)}$ arithmetic operations in K, where two $n \times n$ matrices are multiplied using $O(n^{\omega})$ operations. Previous algorithms required time $(n^2 d)^{1+o(1)}$.

The resultant is the determinant of the Sylvester matrix $S(x)$ of p and q, which is an $n \times n$ Toeplitz-like polynomial matrix of degree d. We use a blocking technique and exploit the structure of $S(x)$ for reducing the determinant computation to the computation of a matrix fraction description $R(x)Q(x)^{-1}$ of an $m \times m$ submatrix of the inverse $S(x)^{-1}$, where $m \ll n$. We rely on fast algorithms for handling dense polynomial matrices: the fraction description is obtained from an x-adic expansion via matrix fraction reconstruction, and the resultant as the determinant of the denominator matrix.

We also describe some extensions of the approach to the computation of generic Gröbner bases and of characteristic polynomials of generic structured matrices and in univariate quotient algebras.

ACM Reference Format:
Gilles Villard. 2018. On Computing the Resultant of Generic Bivariate Polynomials. In *ISSAC'18: 2018 ACM International Symposium on Symbolic and Algebraic Computation, July 16–19, 2018, New York, NY, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3208976.3209020

1 INTRODUCTION

details and references. More precisely, on the one hand, the resultant of two univariate polynomials of degree n (taking $d = 0$ in above definition) can be computed in $O(\mathsf{M}(n) \log n)$ arithmetic operations in K using the Knuth-Schönhage-Moenck algorithm. We use $\mathsf{M}(n)$ for a multiplication time for univariate polynomials of degree bounded by n over K (see for instance [16, Chap. 8]). On the other hand, in our case the resultant has degree at most $2nd$, hence an extra factor nd appears for the evaluation-interpolation cost. In total, it can be shown that the bivariate resultant can be computed using $O(n\,\mathsf{M}(nd) \log(nd))$ arithmetic operations [16, Chap. 11], which is $(n^2 d)^{1+o(1)}$ using $\mathsf{M}(n) = O(n \log n \log\log n)$ with Cantor and Kaltofen's polynomial multiplication [9].

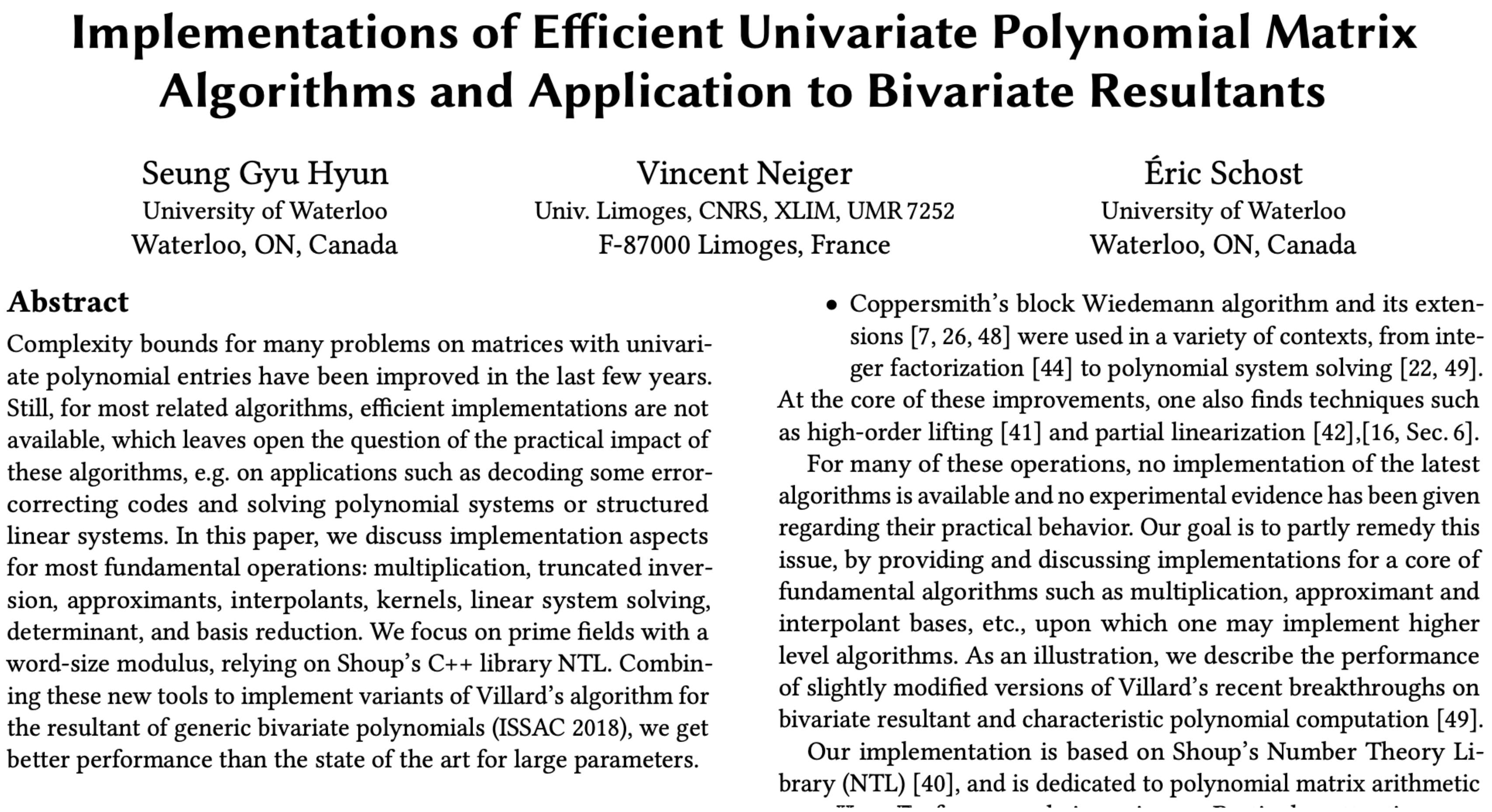
Before giving an overview of our approach let us mention some important results that have been obtained since the initial results cited above. For comprehensive presentations of the resultant and subresultant problem, and detailed history and complexity analyses, the reader may refer to [16, 17, 36]. Especially for avoiding modular methods over $\mathbb{Z}$, recursive subresultant formulas have been given in [17, 38, 43] that allow half-gcd schemes for computing the resultant of polynomials in $D[y]$ where D is a domain such that the exact division can be performed.

The complexity bound $(n^2 d)^{1+o(1)}$ has not been improved in the general case. In some special cases much better complexity bounds are known [5, Sec. 5]. In particular, for univariate f and g of degree n in $K[y]$, the composed sum $(f \oplus g)(x) = \mathrm{Res}_y(f(x-y), g(y))$ and the composed product $(f \otimes g)(x) = \mathrm{Res}_y(y^n f(x/y), g(y))$ can be computed using $n^{2+o(1)}$ operations in K [5]. (The restrictions in [5]

[Villard, ISSAC 2018]

▷ $\mathrm{Res}_y(P(x,y), Q(x,y))$ of *generic* P, Q of degree d in $\tilde{O}(d^{3-1/\omega})$

3. Resultant of Generic Bivariate Polynomials



Implementations of Efficient Univariate Polynomial Matrix Algorithms and Application to Bivariate Resultants

Seung Gyu Hyun
University of Waterloo
Waterloo, ON, Canada

Vincent Neiger
Univ. Limoges, CNRS, XLIM, UMR 7252
F-87000 Limoges, France

Éric Schost
University of Waterloo
Waterloo, ON, Canada

Abstract

Complexity bounds for many problems on matrices with univariate polynomial entries have been improved in the last few years. Still, for most related algorithms, efficient implementations are not available, which leaves open the question of the practical impact of these algorithms, e.g. on applications such as decoding some error-correcting codes and solving polynomial systems or structured linear systems. In this paper, we discuss implementation aspects for most fundamental operations: multiplication, truncated inversion, approximants, interpolants, kernels, linear system solving, determinant, and basis reduction. We focus on prime fields with a word-size modulus, relying on Shoup's C++ library NTL. Combining these new tools to implement variants of Villard's algorithm for the resultant of generic bivariate polynomials (ISSAC 2018), we get better performance than the state of the art for large parameters.

- Coppersmith's block Wiedemann algorithm and its extensions [7, 26, 48] were used in a variety of contexts, from integer factorization [44] to polynomial system solving [22, 49].

At the core of these improvements, one also finds techniques such as high-order lifting [41] and partial linearization [42],[16, Sec. 6].

For many of these operations, no implementation of the latest algorithms is available and no experimental evidence has been given regarding their practical behavior. Our goal is to partly remedy this issue, by providing and discussing implementations for a core of fundamental algorithms such as multiplication, approximant and interpolant bases, etc., upon which one may implement higher level algorithms. As an illustration, we describe the performance of slightly modified versions of Villard's recent breakthroughs on bivariate resultant and characteristic polynomial computation [49].

Our implementation is based on Shoup's Number Theory Library (NTL) [40], and is dedicated to polynomial matrix arithmetic

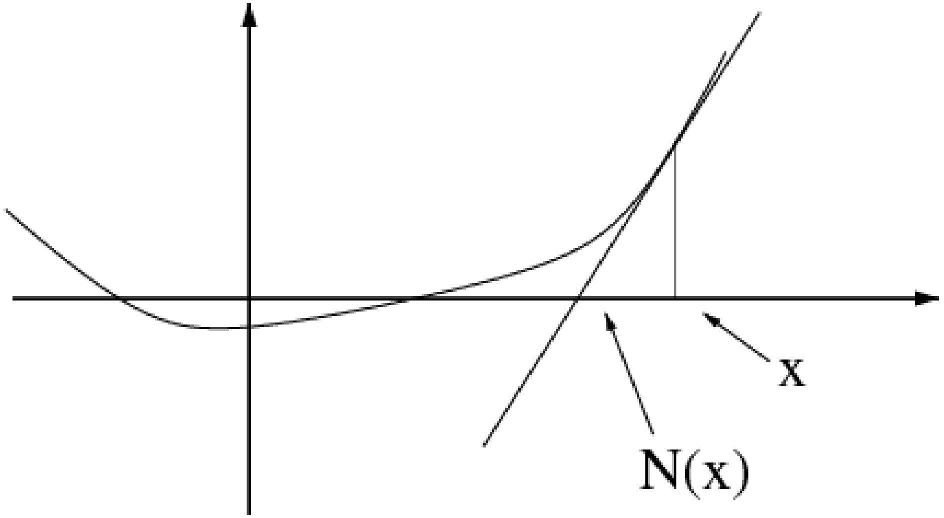
[Hyun, Neiger, Schost, ISSAC 2019]

▷ efficient implementations of (variants) of Villard's 2018 algorithm

Newton Iteration

Newton’s tangent method: real case

[Newton, 1671]



$$x_{\kappa+1} = \mathcal{N}(x_\kappa) = x_\kappa - (x_\kappa^2 - 2)/(2x_\kappa), \quad x_0 = 1$$

$$x_1 = 1.5000000000000000000000000000000$$

$$x_2 = 1.4166666666666666666666666666667$$

$$x_3 = 1.4142156862745098039215686274510$$

$$x_4 = 1.4142135623746899106262955788901$$

$$x_5 = 1.4142135623730950488016896235025$$

Newton's tangent method: power series case

$$x_{\kappa+1} = \mathcal{N}(x_\kappa) = x_\kappa - (x_\kappa^2 - (1-t))/(2x_\kappa), \quad x_0 = 1$$

$$x_1 = 1 - \frac{1}{2}t$$

$$x_2 = 1 - \frac{1}{2}t - \frac{1}{8}t^2 - \frac{1}{16}t^3 - \frac{1}{32}t^4 - \frac{1}{64}t^5 - \frac{1}{128}t^6 - \frac{1}{256}t^7 - \frac{1}{512}t^8 - \frac{1}{1024}t^9 + \cdots$$

$$x_3 = 1 - \frac{1}{2}t - \frac{1}{8}t^2 - \frac{1}{16}t^3 - \frac{5}{128}t^4 - \frac{7}{256}t^5 - \frac{21}{1024}t^6 - \frac{33}{2048}t^7 - \frac{107}{8192}t^8 - \frac{177}{16384}t^9 +$$

Newton's tangent method: power series case

In order to solve $\varphi(x,g)=0$ in $\mathbb{K}[[x]]$ (where $\varphi\in\mathbb{K}[[x,y]]$, $\varphi(0,0)=0$ and $\varphi_y(0,0)\neq 0$), iterate

$$g_{\kappa+1}=g_\kappa-\frac{\varphi(g_\kappa)}{\varphi_y(g_\kappa)}\mod x^{2^{\kappa+1}}$$

$$g-g_{\kappa+1}=g-g_\kappa+\frac{\varphi(g)+(g_\kappa-g)\varphi_y(g)+O((g-g_\kappa)^2)}{\varphi_y(g)+O(g-g_\kappa)}=O((g-g_\kappa)^2).$$

- The number of correct coefficients doubles after each iteration
- Total cost $=$ $\mathbf{2}\times$ $\Big($ the cost of the last iteration $\Big)$

Theorem [Cook 1966, Sieveking 1972 & Kung 1974, Brent 1975]
Division, logarithm and exponential of power series in $\mathbb{K}[[x]]$ can be computed at precision N using $O(\mathsf{M}(N))$ operations in $\mathbb{K}$

Division and logarithm of power series

[Sieveking-Kung, 1972]

To compute the reciprocal of $f \in \mathbb{K}[[x]]$, choose $\varphi(g) = 1/g - f$:

$$g_0 = \frac{1}{f_0} \quad \text{and} \quad g_{\kappa+1} = g_\kappa + g_\kappa(1 - f g_\kappa) \mod x^{2^{\kappa+1}} \quad \text{for } \kappa \geq 0$$

Master Theorem: $\mathsf{C}(N) = \mathsf{C}(N/2) + O(\mathsf{M}(N)) \implies \mathsf{C}(N) = O(\mathsf{M}(N))$

Corollary: division of power series at precision N in $O(\mathsf{M}(N))$

Corollary: Logarithm $\log(f) = -\sum_{i\geq 1} \frac{(1-f)^i}{i}$ of $f \in 1 + x\mathbb{K}[[x]]$ in $O(\mathsf{M}(N))$:

- compute the Taylor expansion of $h = f'/f$ modulo x^{N-1} $O(\mathsf{M}(N))$
- take the antiderivative of h $O(N)$

Details on power series inversion

Lemma Given $F \in \mathbb{K}[[x]]$ with $F(0) \neq 0$, $n \in \mathbb{N}_{>0}$, and $G \in \mathbb{K}[[x]]$ s.t. $G - F^{-1} = O(x^n)$, then $\mathcal{N}(G) = 2G - GFG$ satisfies $\mathcal{N}(G) - F^{-1} = O(x^{2n})$.

Proof: Writing $1 - GF = x^n H$, then inverting $F = G^{-1}(1 - x^n H)$ yields

$$F^{-1} = (1 + x^n H + O(x^{2n}))G = \mathcal{N}(G) + O(x^{2n}).$$

Algorithm (series inversion by Newton iteration)

Input Truncation T to order $N \in \mathbb{N}_{>0}$ of a series $F \in \mathbb{K}[[x]]$ with $F(0) \neq 0$.

Output The truncation S to order N of the inverse series F^{-1}.

```
If $N = 1$, return $T(0)^{-1}$. Otherwise:
1. Recursively compute the truncation $G$ to order $\lceil N/2 \rceil$ of $T^{-1}$.
2. Return $S := G + \mathrm{rem}((1 - GT)G, x^N)$.
```

Details on power series inversion

Algorithm (series inversion by Newton iteration)

Input Truncation T to order $N \in \mathbb{N}_{>0}$ of a series $F \in \mathbb{K}[[x]]$ with $F(0) \neq 0$.

Output The truncation S to order N of the inverse series F^{-1}.

```
If $N = 1$, return $T(0)^{-1}$. Otherwise:
1. Recursively compute the truncation $G$ to order $\lceil N/2 \rceil$ of $T^{-1}$.
2. Return $S := G + \operatorname{rem}((1 - GT)G, x^N)$.
```

Correctness proof Assume $T^{-1} = G + O(x^{\lceil N/2 \rceil})$ by induction. By Lemma,

$$\mathcal{N}(G) - T^{-1} = O(x^{2\lceil N/2 \rceil}) = O(x^N).$$

Write $F = T + O(x^N) = T(1 + O(x^N))$ to observe $F^{-1} = T^{-1} + O(x^N)$. Then,

$$F^{-1} - S = (F^{-1} - T^{-1}) + (T^{-1} - \mathcal{N}(G)) + (\mathcal{N}(G) - S) = O(x^N).$$

Application: Euclidean division for polynomials

[Strassen, 1973]

Pb: Given $F, G \in \mathbb{K}[x]_{\leq N}$, compute (Q, R) in Euclidean division $F = QG + R$

Naive algorithm: $O(N^2)$

Idea: look at $F = QG + R$ from infinity: $Q \sim_{+\infty} F/G$

Let $N = \deg(F)$ and $n = \deg(G)$. Then $\deg(Q) = N - n$, $\deg(R) < n$ and

$$\underbrace{F(1/x)x^N}_{\mathrm{rev}(F)} = \underbrace{G(1/x)x^n}_{\mathrm{rev}(G)} \cdot \underbrace{Q(1/x)x^{N-n}}_{\mathrm{rev}(Q)} + \underbrace{R(1/x)x^{\deg(R)}}_{\mathrm{rev}(R)} \cdot x^{N-\deg(R)}$$

Algorithm:

- Compute $\mathrm{rev}(Q) = \mathrm{rev}(F)/\mathrm{rev}(G) \mod x^{N-n+1}$ $O(\mathsf{M}(N))$
- Recover Q $O(N)$
- Deduce $R = F - QG$ $O(\mathsf{M}(N))$

Exponentials of power series and 1st order LDE

[Brent, 1975]

To compute the exponential $\exp(f) = \sum_{i \geq 0} \frac{f^i}{i!}$, choose $\varphi(g) = \log(g) - f$:

$$g_0 = 1 \quad \text{and} \quad g_{\kappa+1} = g_\kappa - g_\kappa \left(\log(g_\kappa) - f\right) \mod x^{2^{\kappa+1}} \quad \text{for } \kappa \geq 0.$$

Complexity: $\mathsf{C}(N) = \mathsf{C}(N/2) + O(\mathsf{M}(N)) \quad \Longrightarrow \quad \mathsf{C}(N) = O(\mathsf{M}(N))$

Corollary: Solve first order linear differential equations $af' + bf = c$ in $O(\mathsf{M}(N))$

- if $c = 0$ then the solution is $f_0 = \exp\left(-\int b/a\right)$ $O(\mathsf{M}(N))$
- else, variation of constants: $f = f_0 g$, where $g' = c/(af_0)$ $O(\mathsf{M}(N))$

▶ main difficulty for higher orders: for non-commutativity reasons, the matrix exponential $Y(x) = \exp(\int A(x))$ is not a solution of $Y' = A(x)Y$.

Application: conversion coefficients $\leftrightarrow$ power sums

[Schönhage, 1982]

Any polynomial $F = x^n + a_1 x^{n-1} + \cdots + a_n$ in $\mathbb{K}[x]$ can be represented by its first n power sums $S_i = \sum_{F(\alpha)=0} \alpha^i$

Conversions coefficients $\leftrightarrow$ power sums can be performed

- either in $O(n^2)$ using Newton identities (naive way):

$$i a_i + S_1 a_{i-1} + \cdots + S_i = 0, \quad 1 \leq i \leq n$$

- or in $O(\mathsf{M}(n))$ using generating series

$$\frac{\mathsf{rev}(F)'}{\mathsf{rev}(F)} = -\sum_{i \geq 0} S_{i+1} x^i \quad \Longleftrightarrow \quad \mathsf{rev}(F) = \exp\left(-\sum_{i \geq 1} \frac{S_i}{i} x^i\right)$$

A second exercise for next Tuesday

(2) Assume that $F \in \mathbb{K}[[x]]$ with $F(0) = 1$.

(a) What is the complexity of computing $\sqrt{F}$, by using $\sqrt{F} = \exp(\frac{1}{2} \log F)$?

(b) Describe a Newton iteration that directly computes $\sqrt{F}$, without appealing to successive logarithm and exponential computations.

(c) Estimate the complexity of the algorithm in (b).