Vincent Neiger
LIP6, Sorbonne Université, France

designing fast Guruswami-Sudan decoders using univariate polynomial matrix algorithms

CAIPI symposium @ Bordeaux
November 9, 2023

outline

- computer algebra
- Reed-Solomon decoding
- polynomial matrices
- efficient list decoding

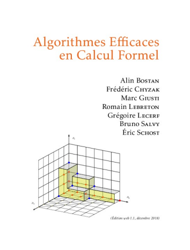
outline

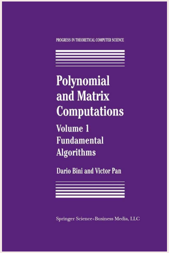
- computer algebra
- Reed-Solomon decoding
- polynomial matrices
- efficient list decoding

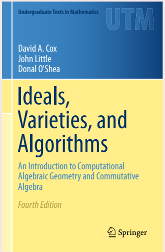
- efficient algorithms and software
- for matrices over a field
- for univariate polynomials

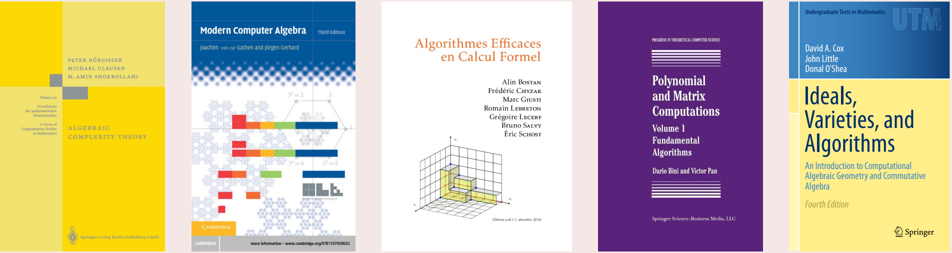

PETER BÜRGISSER
MICHAEL CLAUSEN
M. AMIN SHOKROLLAHI
ALGEBRAIC
COMPLEXITY THEORY
Springer-Verlag Berlin Heidelberg GmbH


Modern Computer Algebra
Third Edition
Joachim von zur Gathen and Jürgen Gerhard
CAMBRIDGE
more information – www.cambridge.org/9781107039032


Algorithmes Efficaces
en Calcul Formel
Alin Bostan
Frédéric Chyzak
Marc Giusti
Romain Lebreton
Grégoire Lecerf
Bruno Salvy
Éric Schost


PROGRESS IN THEORETICAL COMPUTER SCIENCE
Polynomial
and Matrix
Computations
Volume 1
Fundamental
Algorithms
Dario Bini and Victor Pan
Springer Science+Business Media, LLC


Undergraduate Texts in Mathematics
UTM
David A. Cox
John Little
Donal O'Shea
Ideals,
Varieties, and
Algorithms
An Introduction to Computational
Algebraic Geometry and Commutative
Algebra
Fourth Edition
Springer



MAGMA COMPUTER • ALGEBRA

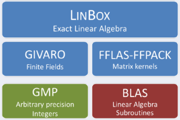
MATLAB®

PARI GP

Maple™

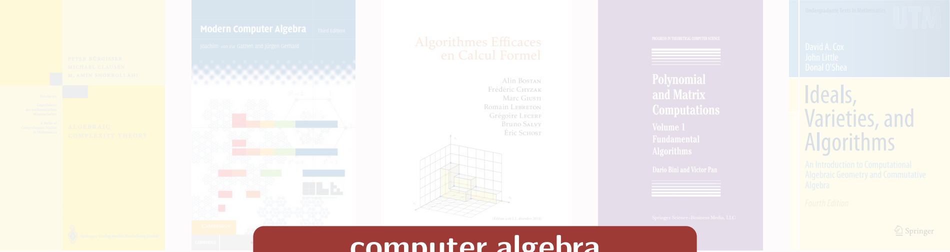
WOLFRAM MATHEMATICA 12

sage



SymPy

CoCoA



computer algebra

algorithm design
and **software implementations**
for **exact computations**
with **mathematical objects**

Euclid's GCD -300

computer algebra

algorithm design
and **software implementations**
for **exact computations**
with **mathematical objects**

Euclid's GCD -300

Gaussian elimination 179

computer algebra

algorithm design
and **software implementations**
for **exact computations**
with **mathematical objects**

Euclid's GCD -300

Gaussian elimination 179

Newton's method 1669

computer algebra

algorithm design
and **software implementations**
for **exact computations**
with **mathematical objects**

Euclid's GCD -300

Gaussian elimination 179

Newton's method 1669

computer algebra

algorithm design
and **software implementations**
for **exact computations**
with **mathematical objects**

FFT 1805, '65

Euclid's GCD -300

Gaussian elimination 179

Newton's method 1669

computer algebra

algorithm design
and **software implementations**
for **exact computations**
with **mathematical objects**

FFT 1805, '65

Karatsuba '62

Euclid's GCD -300

Gaussian elimination 179

Newton's method 1669

computer algebra

algorithm design
and **software implementations**
for **exact computations**
with **mathematical objects**

FFT 1805, '65

Strassen '69

Karatsuba '62

Euclid's GCD -300

Gaussian elimination 179

Newton's method 1669

computer algebra

algorithm design
and **software implementations**
for **exact computations**
with **mathematical objects**

FFT 1805, '65

Buchberger '76

Strassen '69

Karatsuba '62

Euclid's GCD -300

Gaussian elimination 179

Newton's method 1669

computer algebra

algorithm design
and **software implementations**
for **exact computations**
with **mathematical objects**

LLL '82, NFS '88

FFT 1805, '65

Buchberger '76

Strassen '69

Karatsuba '62

Euclid's GCD -300

Gaussian elimination 179

Newton's method 1669

FFT 1805, '65

Karatsuba '62

Principal Discoveries of Efficient Methods of Computing the DFT

Researcher(s)	Date	Sequence Lengths	Number of DFT Values	Application
C. F. Gauss [10]	1805	Any composite integer	All	Interpolation of orbits of celestial bodies
F. Carlini [28]	1828	12	—	Harmonic analysis of barometric pressure
A. Smith [25]	1846	4, 8, 16, 32	5 or 9	Correcting deviations in compasses on ships
J. D. Everett [23]	1860	12	5	Modeling underground temperature deviations
C. Runge [7]	1903	$2^n k$	All	Harmonic analysis of functions
K. Stumpff [16]	1939	$2^n k$, $3^n k$	All	Harmonic analysis of functions
Danielson and Lanczos [5]	1942	2^n	All	X-ray diffraction in crystals
L. H. Thomas [13]	1948	Any integer with relatively prime factors	All	Harmonic analysis of functions
I. J. Good [3]	1958	Any integer with relatively prime factors	All	Harmonic analysis of functions
Cooley and Tukey [1]	1965	Any composite integer	All	Harmonic analysis of functions
S. Winograd [14]	1976	Any integer with relatively prime factors	All	Use of complexity theory for harmonic analysis

Euclid's GCD -300

Gaussian elimination 179

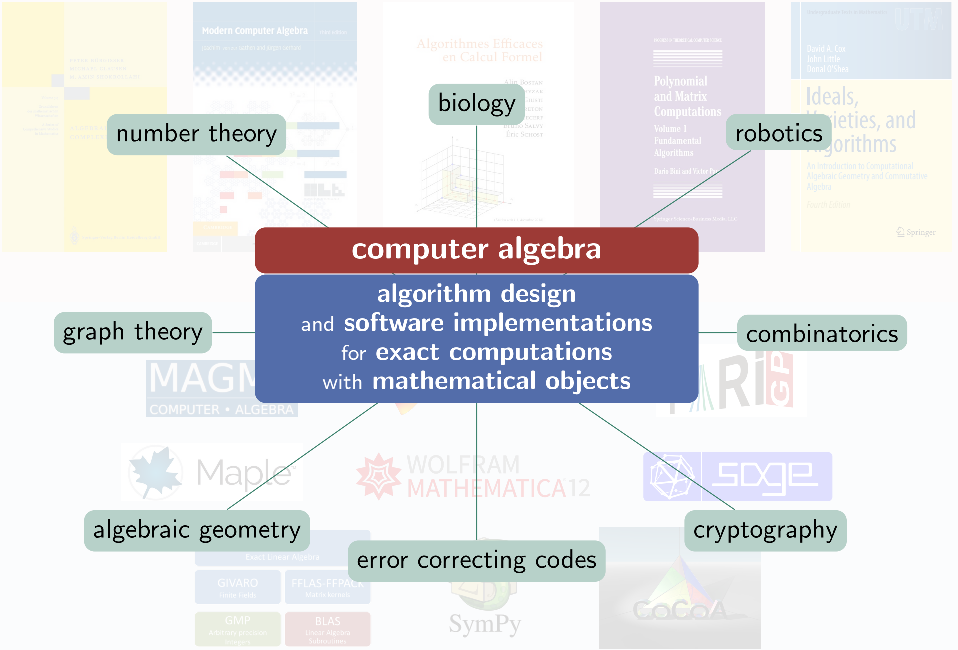
Newton's method 1669

Principal Discoveries of Efficient Methods of Computing the DFT

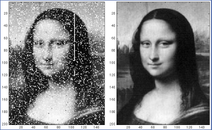
Researcher(s)	Date	Sequence Lengths	Number of DFT Values	Application
C. F. Gauss [10]	1805	Any composite integer	All	Interpolation of orbits of celestial bodies
F. Carlini [28]	1828	12	—	Harmonic analysis of barometric pressure
A. Smith [25]	1846	4, 8, 16, 32	5 or 9	Correcting deviations in compasses on ships
J. D. Everett [23]	1860	12	5	Modeling underground temperature deviations
C. Runge [7]	1903	$2^n k$	All	Harmonic analysis of functions
K. Stumpff [16]	1939	$2^n k$, $3^n k$	All	Harmonic analysis of functions
Danielson and Lanczos [5]	1942	2^n	All	X-ray diffraction in crystals
L. H. Thomas [13]	1948	Any integer with relatively prime factors	All	Harmonic analysis of functions
I. J. Good [3]	1958	Any integer with relatively prime factors	All	Harmonic analysis of functions
Cooley and Tukey [1]	1965	Any composite integer	All	Harmonic analysis of functions
S. Winograd [14]	1976	Any integer with relatively prime factors	All	Use of complexity theory for harmonic analysis

FFT 1805, '65

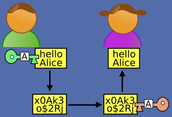
Karatsuba '62


computer algebra
algorithm design
and software implementations
for exact computations
with mathematical objects
number theory
biology
robotics
graph theory
combinatorics
algebraic geometry
error correcting codes
cryptography

error correcting codes



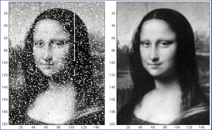
cryptographic protocols



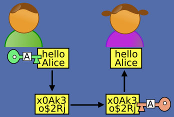
XXth-XXIst centuries : digital data & interconnected networks

integrity – confidentiality

discrete structures : exact and intensive computations error correcting codes



cryptographic protocols



XXth-XXIst centuries : digital data & interconnected networks

integrity – confidentiality

discrete structures : exact and intensive computations

- matrices of large size, with sparsity or structure
- polynomials and polynomial matrices in one variable
- polynomials in several variables

goal of computer algebra

fast algorithms : complexity & efficient implementations

reduce to efficient building blocks

- `MatMul`: matrix multiplication
- `PolMul`: polynomial multiplication

measuring efficiency

efficient algorithms for polynomials, matrices, power series, ...
with coefficients in some base field $\mathbb{K}$

- low **complexity bound**
- low **execution time**

low memory usage, power consumption, ...

prime field $\mathbb{F}_p = \mathbb{Z}/p\mathbb{Z}$
field extension $\mathbb{F}_p[x]/\langle f(x) \rangle$
rational numbers $\mathbb{Q}$

measuring efficiency

efficient algorithms for polynomials, matrices, power series, ...
with coefficients in some base field $\mathbb{K}$

- low **complexity bound**
- low **execution time**

low memory usage, power consumption, ...

prime field $\mathbb{F}_p = \mathbb{Z}/p\mathbb{Z}$
field extension $\mathbb{F}_p[x]/\langle f(x) \rangle$
rational numbers $\mathbb{Q}$

algebraic complexity bounds
$\rightsquigarrow$ count number of operations in $\mathbb{K}$

- 👍 standard complexity model for algebraic computations
- 👍 accurate for finite fields $\mathbb{K} = \mathbb{F}_p$
- 👎 ignores coefficient growth, e.g. over $\mathbb{K} = \mathbb{Q}$

measuring efficiency

efficient algorithms for polynomials, matrices, power series, ...
with coefficients in some base field $\mathbb{K}$

- low **complexity bound**
- low **execution time**

low memory usage, power consumption, ...

prime field $\mathbb{F}_p = \mathbb{Z}/p\mathbb{Z}$
field extension $\mathbb{F}_p[x]/\langle f(x) \rangle$
rational numbers $\mathbb{Q}$

practical performance
$\rightsquigarrow$ measure software running time

this talk:

- working over $\mathbb{K} = \mathbb{F}_p$ with word-size prime p
- Intel Core i7-7600U @ 2.80GHz, no multithreading

matrices: multiplication

$$\mathbf{M} = \begin{bmatrix} 28 & 68 & 75 & 70 \\ 38 & 25 & 75 & 55 \\ 24 & 1 & 56 & 28 \end{bmatrix} \in \mathbb{K}^{3\times 4} \longrightarrow 3 \times 4 \text{ matrix over } \mathbb{K} \text{ (here } \mathbb{F}_{97}\text{)}$$

fundamental operations on $m \times m$ matrices:

- addition is "quadratic": $O(m^2)$ operations in $\mathbb{K}$
- naive multiplication is cubic: $O(m^3)$

[Strassen'69]

breakthrough: **subcubic** matrix multiplication

matrices: multiplication

$$\mathbf{M} = \begin{bmatrix} 28 & 68 & 75 & 70 \\ 38 & 25 & 75 & 55 \\ 24 & 1 & 56 & 28 \end{bmatrix} \in \mathbb{K}^{3\times 4} \longrightarrow 3 \times 4 \text{ matrix over } \mathbb{K} \text{ (here } \mathbb{F}_{97})$$

fundamental operations on $m \times m$ matrices:

- addition is "quadratic": $O(m^2)$ operations in $\mathbb{K}$
- naive multiplication is cubic: $O(m^3)$

[Strassen'69]

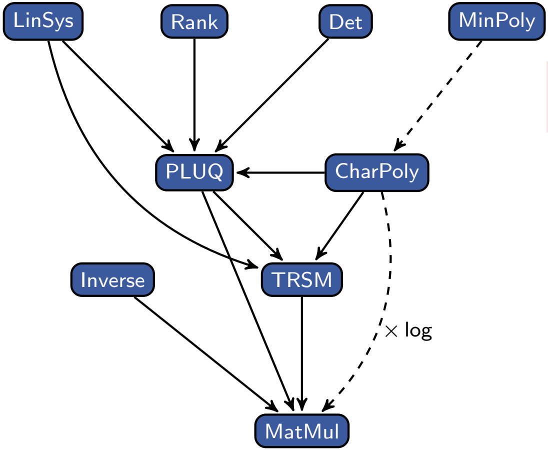
breakthrough: **subcubic** matrix multiplication

- complexity exponent $\omega \approx 2.81$ — i.e. $O(m^{\omega})$ complexity
- used in practice for $m \geqslant$ a few 100s
  in NTL, FLINT, fflas-ffpack...

- best-known exponent $\omega \approx 2.373$
  [Le Gall'14] [Alman-Williams'20]
- "galactic" algorithms: strongly impractical as such

matrices: main computational problems

reductions of most problems to matrix multiplication



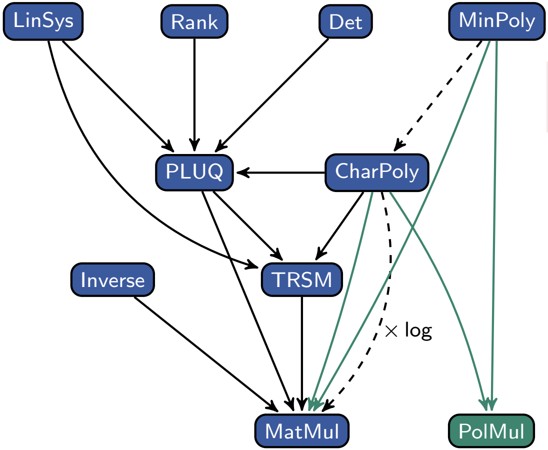
PLUQ = Gaussian elimination
TRSM = triangular solving

LinSys, Det, Rank, PLUQ, TRSM, Inverse $\Big\}$ = O(MatMul)

not closed:
open:

matrices: main computational problems

reductions of most problems to matrix multiplication



PLUQ = Gaussian elimination
TRSM = triangular solving

LinSys, Det, Rank, PLUQ, TRSM, Inverse $\Big\} = O(\text{MatMul})$

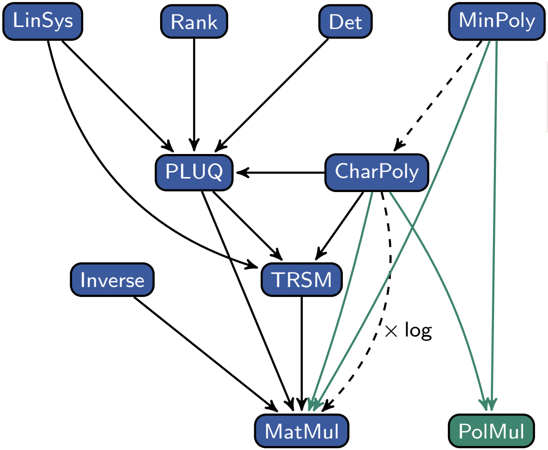
MinPoly, CharPoly $\Big\} = O(\text{MatMul})$

exploiting **non-naive** PolMul

not closed:
open:

matrices: main computational problems

reductions of most problems to matrix multiplication



PLUQ = Gaussian elimination
TRSM = triangular solving

$$\left.\begin{array}{c}\text{LinSys}\\ \text{Det}\\ \text{Rank}\\ \text{PLUQ}\\ \text{TRSM}\\ \text{Inverse}\end{array}\right\} = O(\text{MatMul})$$

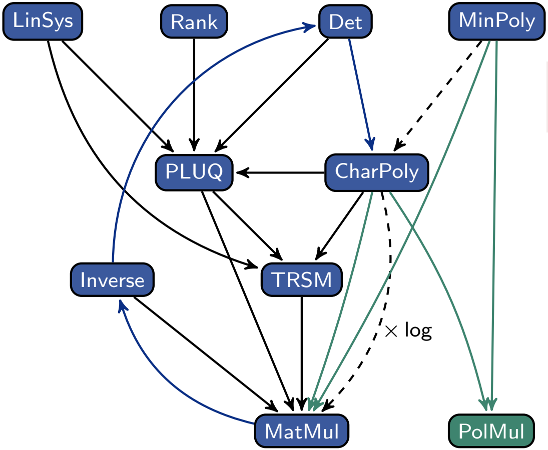
$$\left.\begin{array}{c}\text{MinPoly}\\ \text{CharPoly}\end{array}\right\} = O(\text{MatMul})$$

exploiting **non-naive** PolMul

not closed: is Frobenius normal form in O(MatMul)?
open:

matrices: main computational problems

reductions of most problems to matrix multiplication



PLUQ = Gaussian elimination
TRSM = triangular solving

LinSys, Det, Rank, PLUQ, TRSM, Inverse } = O(MatMul)

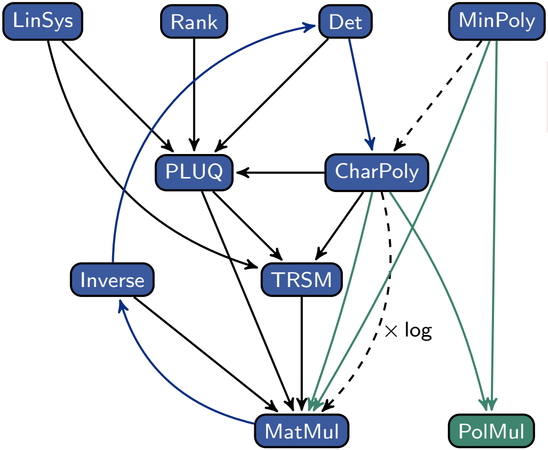
MinPoly, CharPoly } = O(MatMul)

exploiting **non-naive** PolMul

not closed: is Frobenius normal form in O(MatMul)?
open:

matrices: main computational problems

reductions of most problems to matrix multiplication



PLUQ = Gaussian elimination
TRSM = triangular solving

LinSys, Det, Rank, PLUQ, TRSM, Inverse $\Big\}$ = O(MatMul)

MinPoly, CharPoly $\Big\}$ = O(MatMul)

exploiting **non-naive** PolMul

not closed: is Frobenius normal form in O(MatMul)?
open: is linear system solving as hard as multiplication?

bonus: some notes

biblio: `https://www.sciencedirect.com/science/article/pii/S0747717113000631`

- explicit reductions between inversion & MatMul & variants of Gaussian elimination / echelon form computation
- constants in the $O(\cdot)$ complexities when using classical matrix multiplication ($\omega = 3$) or Strassen's algorithm

"not closed": it is open, but

- there is a randomized algorithm for Frobenius form computation which has complexity $O(\text{MatMul})$
  $\rightsquigarrow$ `http://www.cs.uwaterloo.ca/~astorjoh/cpoly.pdf`
- recent developments for the characteristic polynomial gives new insight concerning core operations typically used in Frobenius form algorithms

polynomials: multiplication

$p = 87x^7 + 74x^6 + 60x^5 + 46x^4 + 16x^3 + 41x^2 + 86x + 69$

$p \in \mathbb{K}[x]_{<8}$ $\longrightarrow$ univariate polynomial in x of degree < 8 over $\mathbb{K}$

fundamental operations on polynomials of degree $< d$:

- addition and Horner's evaluation are linear: $O(d)$
- naive multiplication is quadratic: $O(d^2)$

[Karatsuba'62] $M(d) \in O(d^{1.58})$

breakthrough: **subquadratic** polynomial multiplication

polynomials: multiplication

$p = 87x^7 + 74x^6 + 60x^5 + 46x^4 + 16x^3 + 41x^2 + 86x + 69$

$p \in \mathbb{K}[x]_{<8}$ $\longrightarrow$ univariate polynomial in x of degree < 8 over $\mathbb{K}$

fundamental operations on polynomials of degree $< d$:

- addition and Horner's evaluation are linear: $O(d)$
- naive multiplication is quadratic: $O(d^2)$

[Karatsuba'62] $M(d) \in O(d^{1.58})$

breakthrough: **subquadratic** polynomial multiplication

[Schönhage-Strassen'71] [Nussbaumer'80] [Cantor-Kaltofen'91] $M(d) \in O(d \log(d) \log\log(d))$

breakthrough: **quasi-linear** polynomial multiplication

research still active, with recent progress by [Harvey-van der Hoeven-Lecerf]

- change of representation by evaluation-interpolation
- used in practice as soon as $d \approx 100$
- FFT techniques using (virtual) roots of unity

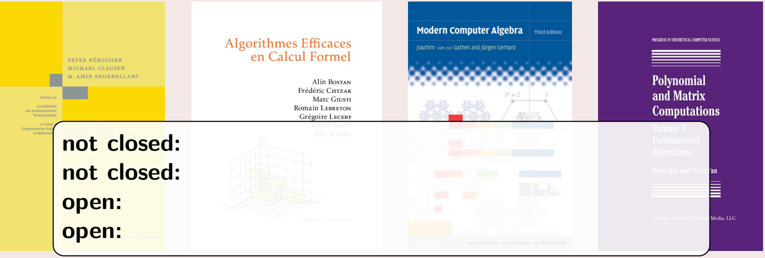
note: $M(d) \in O(d \log(d))$
if provided a "good" root of unity

polynomials: main computational problems

most problems have quasi-linear complexity

thanks to reductions to `PolMul`

- addition $f + g$, multiplication $f * g$
- division with remainder $f = qg + r$
- truncated inverse $f^{-1} \bmod x^d$
- extended GCD $fu + gv = \gcd(f, g)$
- multipoint eval. $f \mapsto f(\alpha_1), \ldots, f(\alpha_d)$
- interpolation $f(\alpha_1), \ldots, f(\alpha_d) \mapsto f$
- Padé approximation $f = \frac{p}{q} \bmod x^d$
- minpoly of linearly recurrent sequence



not closed:
not closed:
open:
open:

polynomials: main computational problems

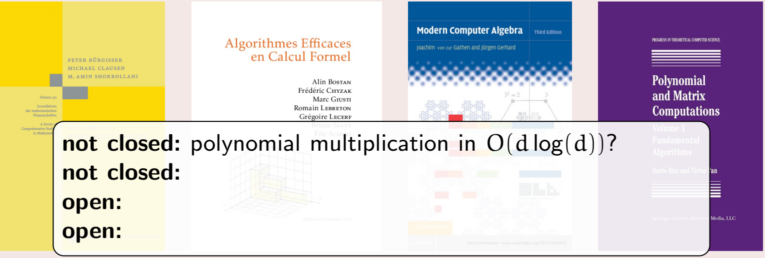
most problems have quasi-linear complexity

thanks to reductions to `PolMul`

$O(M(d))$

- addition $f+g$, multiplication $f*g$
- division with remainder $f = qg + r$
- truncated inverse $f^{-1} \bmod x^d$
- extended GCD $fu + gv = \gcd(f,g)$

- multipoint eval. $f \mapsto f(\alpha_1), \dots, f(\alpha_d)$
- interpolation $f(\alpha_1), \dots, f(\alpha_d) \mapsto f$
- Padé approximation $f = \frac{p}{q} \bmod x^d$
- minpoly of linearly recurrent sequence



not closed: polynomial multiplication in $O(d \log(d))$?
not closed:
open:
open:

polynomials: main computational problems

most problems have quasi-linear complexity

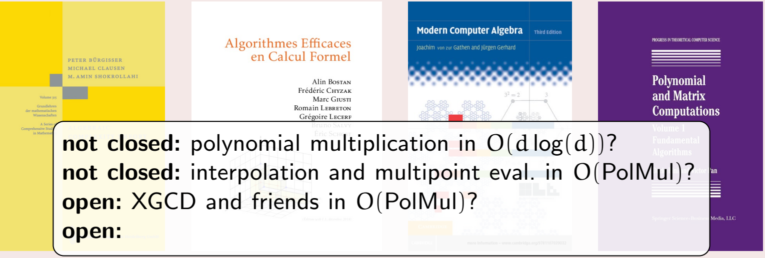
thanks to reductions to `PolMul`

$O(M(d))$

- addition $f + g$, multiplication $f * g$
- division with remainder $f = qg + r$
- truncated inverse $f^{-1} \bmod x^d$
- extended GCD $fu + gv = \gcd(f, g)$

$O(M(d)\log(d))$

- multipoint eval. $f \mapsto f(\alpha_1), \ldots, f(\alpha_d)$
- interpolation $f(\alpha_1), \ldots, f(\alpha_d) \mapsto f$
- Padé approximation $f = \frac{p}{q} \bmod x^d$
- minpoly of linearly recurrent sequence



not closed: polynomial multiplication in $O(d\log(d))$?
not closed: interpolation and multipoint eval. in $O(\mathrm{PolMul})$?
open: XGCD and friends in $O(\mathrm{PolMul})$?
open:

polynomials: main computational problems

most problems have quasi-linear complexity

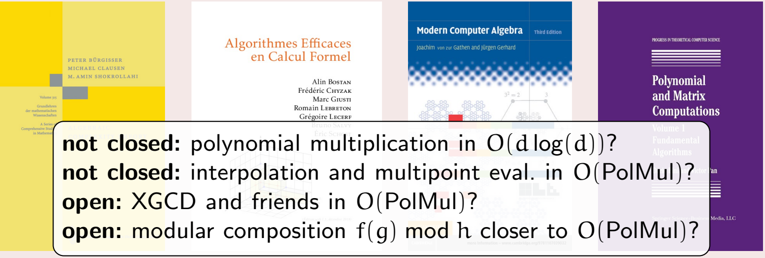
thanks to reductions to `PolMul`

$O(M(d))$

- addition $f+g$, multiplication $f*g$
- division with remainder $f=qg+r$
- truncated inverse $f^{-1} \bmod x^d$
- extended GCD $fu+gv=\gcd(f,g)$

$O(M(d)\log(d))$

- multipoint eval. $f \mapsto f(\alpha_1),\ldots,f(\alpha_d)$
- interpolation $f(\alpha_1),\ldots,f(\alpha_d) \mapsto f$
- Padé approximation $f=\frac{p}{q} \bmod x^d$
- minpoly of linearly recurrent sequence



not closed: polynomial multiplication in $O(d\log(d))$?
not closed: interpolation and multipoint eval. in $O(\mathrm{PolMul})$?
open: XGCD and friends in $O(\mathrm{PolMul})$?
open: modular composition $f(g) \bmod h$ closer to $O(\mathrm{PolMul})$?

bonus: some notes

interpolation and multipoint eval. in $\mathrm{O}(\mathsf{PolMul})$ "not closed":

- remains open for an arbitrary set of points, with no assumption, but:
- by design, solved for FFT points (powers of some root of unity)
- more generally, solved for points forming a geometric sequence
  `https://www.sciencedirect.com/science/article/pii/S0885064X05000026`
- in many applications of interpolation/evaluation, one can choose the points, in which case $\mathrm{O}(\mathsf{PolMul})$ is feasible

polynomial multiplication in $\mathrm{O}(d \log(d))$ "not closed":

- remains open over an arbitrary field, concerning algebraic complexity
- solved when the field possesses suitable roots of unity for FFT
- method of choice in practice (using several primes and CRT if needed) when working over prime finite fields $\mathbb{Z}/p\mathbb{Z}$
- recent progress in the bit complexity model
  `https://www.sciencedirect.com/science/article/pii/S0885064X19300378`
  `https://dl.acm.org/doi/abs/10.1145/3505584`

```
sage: M.degree_matrix(shifts=[-1,2], row_wise=False)
[ 0 -2 -1]
[ 5 -2 -2]
```

hermite_form(*include_zero_rows=True, transformation=False*)

Return the Hermite form of this matrix.

The Hermite form is also normalized, i.e., the pivot polynomials are monic.

INPUT:

- `include_zero_rows` – boolean (default: `True`); if `False`, the zero rows in the output deleted
- `transformation` – boolean (default: `False`); if `True`, return the transformation mat

OUTPUT:

```
// order that remains to be dealt with
VecLong rem_order(order);

// indices of columns/orders that remain to be dealt with
VecLong rem_index(cdim);
std::iota(rem_index.begin(), rem_index.end(), 0);

// all along the algorithm, shift = shifted row degrees of approximant ba
// (initially, input shift = shifted row degree of the identity matrix)

while (not rem_order.empty())
{
    /** Invariant:
     *  - appbas is a shift-ordered weak Popov approximant basis for
     *  (pmat,reached_order) where doneorder is the tuple such that
     *  -->reached_order[j] + rem_order[j] == order[j] for j appearing in
     *  -->reached_order[j] == order[j] for j not appearing in rem_index
     *  - shift == the "input shift"-row degree of appbas
     *  - residual == submatrix of columns (appbas * pmat)[:,j] for all j
```

matrices **software** polynomials

```
sage: M.<x> = GF(7)[]
sage: A = matrix(M, 2, 3, [x, 1, 2*x, x, 1+x, 2])
sage: A.hermite_form()
[      x       1     2*x]
[      0       x 5*x + 2]
sage: A.hermite_form(transformation=True)
(
[      x       1     2*x]  [1 0]
[      0       x 5*x + 2], [6 1]
)
sage: A = matrix(M, 2, 3, [x, 1, 2*x, 2*x, 2, 4*x])
sage: A.hermite_form(transformation=True, include_zero_rows=False)
([  x   1 2*x], [0 4])
sage: H, U = A.hermite_form(transformation=True, include_zero_rows=True); H, U
(
[  x   1 2*x]  [0 4]
[  0   0   0], [5 1]
)
sage: U * A == H
True
sage: H, U = A.hermite_form(transformation=True, include_zero_rows=False)
sage: U * A
[  x   1 2*x]
sage: U * A == H
True
```

See also: `is_hermite()`.

is_hermite(*row_wise=True, lower_echelon=False, include_zero_vectors=True*)

Return a boolean indicating whether this matrix is in Hermite form.

```
            j = std::distance(rem_order.begin(), std::max_element(rem_order.be
);

        long deg = order[rem_index[j]] - rem_order[j];

        // record the coefficients of degree deg of the column j of residual
        // also keep track of which of these are nonzero,
        // and among the nonzero ones, which is the first with smallest shift
        Vec<zz_p> const_residual;
        const_residual.SetLength(rdim);
        VecLong indices_nonzero;
        long piv = -1;
        for (long i = 0; i < rdim; ++i)
        {
            const_residual[i] = coeff(residual[i][j],deg);
            if (const_residual[i] != 0)
            {
                indices_nonzero.push_back(i);
                if (piv<0 || shift[i] < shift[piv])
                    piv = i;
            }
        }

        // if indices_nonzero is empty, const_residual is already zero, there
        if (not indices_nonzero.empty())
        {
            // update all rows of appbas and residual in indices_nonzero exce
```

src/mat_lzz_pX_approximant.cpp

open-source mathematics software system

sage — *Python/Cython*

high-performance exact linear algebra

LinBox – fflas-ffpack *C/C++*

high-performance polynomials (and more)

NTL & FLINT *C/C++*

matrices **software** polynomials

```
sage: M.<x> = GF(7)[]
sage: A = matrix(M, 2, 3, [x, 1, 2*x, x, 1+x, 2])
sage: A.hermite_form()
[      x       1     2*x]
[      0       x 5*x + 2]
sage: A.hermite_form(transformation=True)
(
[      x       1     2*x]  [1 0]
[      0       x 5*x + 2], [6 1]
)
sage: A = matrix(M, 2, 3, [x, 1, 2*x, 2*x, 2, 4*x])
sage: A.hermite_form(transformation=True, include_zero_rows=False)
([  x   1 2*x], [0 4])
sage: H, U = A.hermite_form(transformation=True, include_zero_rows=True); H, U
(
[  x   1 2*x]  [0 4]
[  0   0   0], [5 1]
)
sage: U * A == H
True
sage: H, U = A.hermite_form(transformation=True, include_zero_rows=False)
sage: U * A
[  x   1 2*x]
sage: U * A == H
True
```

See also: is_hermite() .

is_hermite(*row_wise=True, lower_echelon=False, include_zero_vectors=True*)

Return a boolean indicating whether this matrix is in Hermite form.

```
    // order that remains to be dealt with
    VecLong rem_order(order);

    // indices of columns/orders that remain to be dealt with
    VecLong rem_index(cdim);
    std::iota(rem_index.begin(), rem_index.end(), 0);

    // all along the algorithm, shift = shifted row degrees of approximant ba
    // (initially, input shift = shifted row degree of the identity matrix)

    while (not rem_order.empty())
    {
        /** Invariant:
         * - appbas is a shift-ordered weak Popov approximant basis for
         * (pmat,reached_order) where doneorder is the tuple such that
         * -->reached_order[j] + rem_order[j] == order[j] for j appearing in
         * -->reached_order[j] == order[j] for j not appearing in rem_index
         * - shift == the "input shift"-row degree of appbas
         * - residual == submatrix of columns (appbas * pmat)[:,j] for all j

            j = std::distance(rem_order.begin(), std::max_element(rem_order.be
);

        long deg = order[rem_index[j]] - rem_order[j];

        // record the coefficients of degree deg of the column j of residual
        // also keep track of which of these are nonzero,
        // and among the nonzero ones, which is the first with smallest shift
        Vec<zz_p> const_residual;
        const_residual.SetLength(rdim);
        VecLong indices_nonzero;
        long piv = -1;
        for (long i = 0; i < rdim; ++i)
        {
            const_residual[i] = coeff(residual[i][j],deg);
            if (const_residual[i] != 0)
            {
                indices_nonzero.push_back(i);
                if (piv<0 || shift[i] < shift[piv])
                    piv = i;
            }
        }

        // if indices_nonzero is empty, const_residual is already zero, there
        if (not indices_nonzero.empty())
        {
            // update all rows of appbas and residual in indices_nonzero exce
```

src/mat_lzz_pX_approximant.cpp

open-source mathematics software system

sage — *Python/Cython*

high-performance exact linear algebra

LinBox – fflas-ffpack *C/C++*

high-performance polynomials (and more)

NTL & FLINT *C/C++*

- choice of algorithms
- data structures and storage
- cache efficiency
- SIMD vectorization instructions
- multithreading, GPU programming

matrices **software** polynomials

```
sage: M.<x> = GF(7)[]
sage: A = matrix(M, 2, 3, [x, 1, 2*x, x, 1+x, 2])
sage: A.hermite_form()
[      x       1     2*x]
[      0       x 5*x + 2]
sage: A.hermite_form(transformation=True)
(
[      x       1     2*x]  [1 0]
[      0       x 5*x + 2], [6 1]
)
sage: A = matrix(M, 2, 3, [x, 1, 2*x, 2*x, 2, 4*x])
sage: A.hermite_form(transformation=True, include_zero_rows=False)
([  x   1 2*x], [0 4])
sage: H, U = A.hermite_form(transformation=True, include_zero_rows=True); H, U
(
[  x   1 2*x]  [0 4]
[  0   0   0], [5 1]
)
sage: U * A == H
True
sage: H, U = A.hermite_form(transformation=True, include_zero_rows=False)
sage: U * A
[  x   1 2*x]
sage: U * A == H
True
```

See also: is_hermite() .

is_hermite(*row_wise=True, lower_echelon=False, include_zero_vectors=True*)

Return a boolean indicating whether this matrix is in Hermite form.

```
            j = std::distance(rem_order.begin(), std::max_element(rem_order.be
);

        long deg = order[rem_index[j]] - rem_order[j];

        // record the coefficients of degree deg of the column j of residual
        // also keep track of which of these are nonzero,
        // and among the nonzero ones, which is the first with smallest shift
        Vec<zz_p> const_residual;
        const_residual.SetLength(rdim);
        VecLong indices_nonzero;
        long piv = -1;
        for (long i = 0; i < rdim; ++i)
        {
            const_residual[i] = coeff(residual[i][j],deg);
            if (const_residual[i] != 0)
            {
                indices_nonzero.push_back(i);
                if (piv<0 || shift[i] < shift[piv])
                    piv = i;
            }
        }

        // if indices_nonzero is empty, const_residual is already zero, there
        if (not indices_nonzero.empty())
        {
            // update all rows of appbas and residual in indices_nonzero exce
```

src/mat_lzz_pX_approximant.cpp

open-source mathematics software system

sage *Python/Cython*

high-performance exact linear algebra

LinBox – fflas-ffpack $C/C{+}{+}$

high-performance polynomials (and more)

NTL & FLINT $C/C{+}{+}$

- choice of algorithms
- data structures and storage
- cache efficiency
- SIMD vectorization instructions
- multithreading, GPU programming

matrices **software** polynomials

what you can compute in about 1 second

with fflas-ffpack

- `PLUQ` $m = 3800$ 1.00s
- `LinSys` $m = 3800$ 1.00s
- `MatMul` $m = 3000$ 0.97s
- `Inverse` $m = 2800$ 1.01s
- `CharPoly` $m = 2000$ 1.09s

with NTL

- `PolMul` $d = 7 \times 10^6$ 1.03s
- `Division` $d = 4 \times 10^6$ 0.96s
- `XGCD` $d = 2 \times 10^5$ 0.99s
- `MinPoly` $d = 2 \times 10^5$ 1.10s
- `MPeval` $d = 1 \times 10^4$ 1.01s

outline

- computer algebra
  - efficient algorithms and software
  - for matrices over a field
  - for univariate polynomials
- Reed-Solomon decoding
- polynomial matrices
- efficient list decoding

outline

- **computer algebra**
  - efficient algorithms and software
  - for matrices over a field
  - for univariate polynomials
- **Reed-Solomon decoding**
  - context and unique decoding problem
  - key equations and how to solve them
  - correcting more errors?
- **polynomial matrices**
- **efficient list decoding**

error-correcting codes

goal:

reliable data transmission over unreliable communication channel

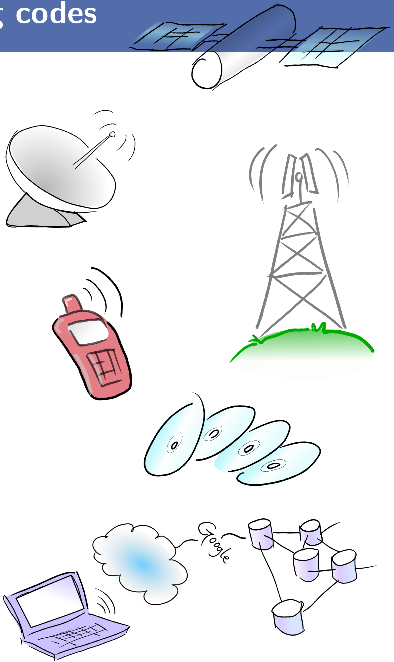
modern development pioneered by Hamming (1940s), Shannon (1948)

strategy:

add redundancy to the message
add redundancy to the message
add redundancy to the message

intended word $(w_0, \ldots, w_k)$ $\longrightarrow$ code word $(c_1, \ldots, c_n)$

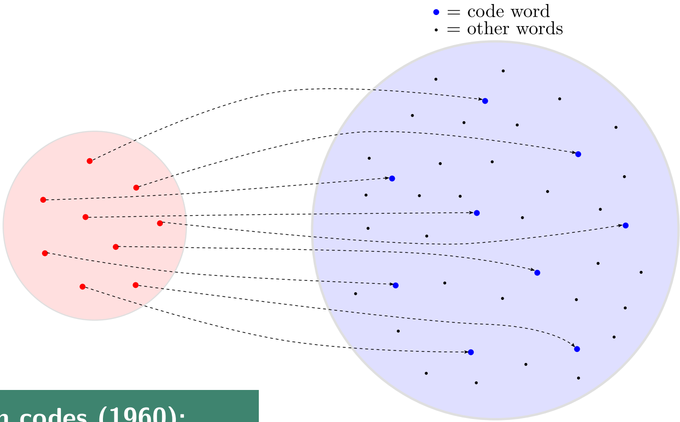
with $\frac{k+1}{n} \leqslant 1$



(drawing: courtesy of Johan Nielsen→Rosenkilde)

encoding: adding redundancy

all intended words $(w_0, \ldots, w_k)$ —encoding→ all code words $(c_1, \ldots, c_n)$



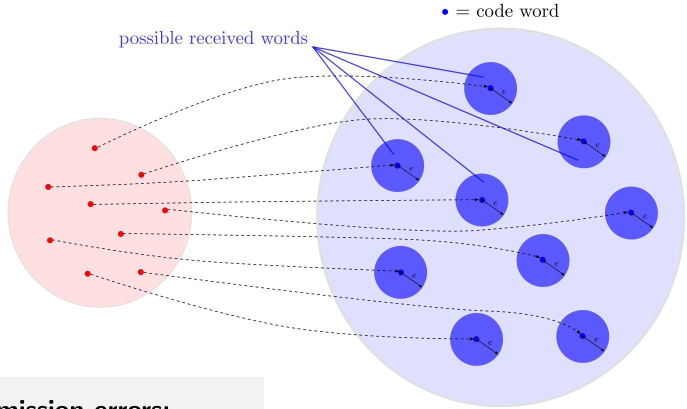
Reed-Solomon codes (1960):

polynomials of degree $\leqslant k$ —encoding→ their evaluations at $\alpha_1, \ldots, \alpha_n$

$w(x) = w_0 + w_1 x + \cdots + w_k x^k$ $\quad\longrightarrow\quad$ $(w(\alpha_1), \ldots, w(\alpha_n))$

transmission over unreliable channel

polynomial $w(x)$ of degree $\leqslant k$ $\xrightarrow{\text{encoding}}$ code word $(w(\alpha_1), \ldots, w(\alpha_n))$ $\dashrightarrow$ noisy channel $\dashrightarrow$ received word $(\beta_1, \ldots, \beta_n)$



noise $\Rightarrow$ transmission errors:

- number of errors $\leqslant e$, meaning $\#\{i \mid w(\alpha_i) \neq \beta_i\} \leqslant e$ (Hamming distance)
- possible received words = balls of radius e centered on the code words

unique decoding

decoding:

find the polynomial $w(x)$ of degree $\leqslant k$
such that $\#\{i \mid w(\alpha_i) \neq \beta_i\} \leqslant e$

- $(\alpha_1, \ldots, \alpha_n)$ = encoding points
- $(\beta_1, \ldots, \beta_n)$ = received word
- $n - e$ = agreement

well-defined:

- existence of w?
- uniqueness of w?

unique decoding

decoding:

find the polynomial $w(x)$ of degree $\leqslant k$ such that $\#\{i \mid w(\alpha_i) \neq \beta_i\} \leqslant e$

- $(\alpha_1, \ldots, \alpha_n)$ = encoding points
- $(\beta_1, \ldots, \beta_n)$ = received word
- $n - e$ = agreement

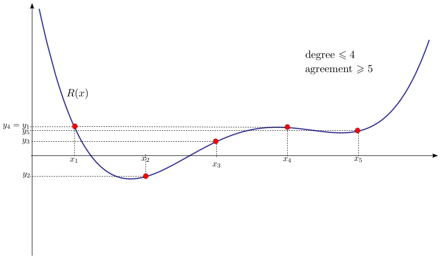
well-defined:

- existence of w?
- uniqueness of w?

$n = 5, k = 4$

$e = 0$: Lagrange interpolation

$e = 1$: no error detection!



unique decoding

decoding:

find the polynomial $w(x)$ of degree $\leqslant k$ such that $\#\{i \mid w(\alpha_i) \neq \beta_i\} \leqslant e$

- $(\alpha_1, \ldots, \alpha_n)$ = encoding points
- $(\beta_1, \ldots, \beta_n)$ = received word
- $n - e$ = agreement

well-defined:

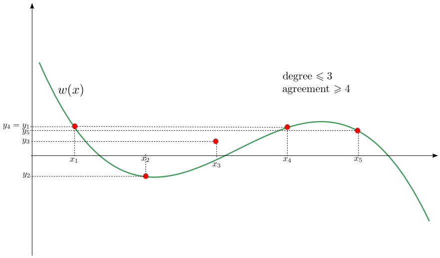
- existence of w?
- uniqueness of w?

$n = 5, k = 3$

$e = 0$: Lagrange interpolant exists!

$e = 1$: up to 5 possible solutions...

→ error is detected, not corrected



unique decoding

decoding:

find the polynomial $w(x)$ of degree $\leqslant k$ such that $\#\{i \mid w(\alpha_i) \neq \beta_i\} \leqslant e$

- $(\alpha_1, \ldots, \alpha_n)$ = encoding points
- $(\beta_1, \ldots, \beta_n)$ = received word
- $n - e$ = agreement

well-defined:

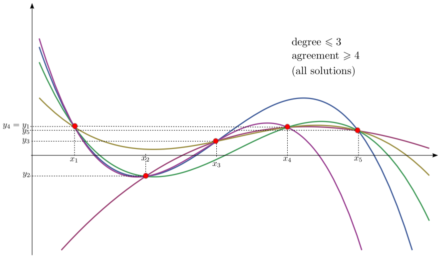
- existence of w?
- uniqueness of w?

$n = 5, k = 3$

$e = 0$: Lagrange interpolant exists!

$e = 1$: up to 5 possible solutions...

$\rightarrow$ error is detected, not corrected



unique decoding

decoding:

find the polynomial $w(x)$ of degree $\leqslant k$ such that $\#\{i \mid w(\alpha_i) \neq \beta_i\} \leqslant e$

. $(\alpha_1, \ldots, \alpha_n)$ = encoding points
. $(\beta_1, \ldots, \beta_n)$ = received word
. $n - e$ = agreement

well-defined:

. existence of w? by construction

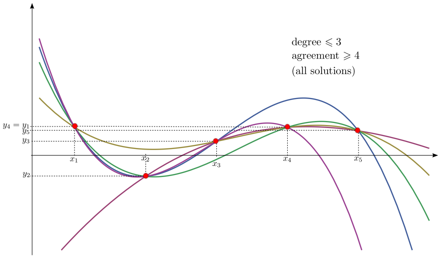
. uniqueness of w? a priori ... yet, guaranteed **if** no overlap between the balls of possible received words

$n = 5, k = 3$

$e = 0$: Lagrange interpolant exists!

$e = 1$: up to 5 possible solutions...

$\rightarrow$ error is detected, not corrected



unique decoding

decoding:

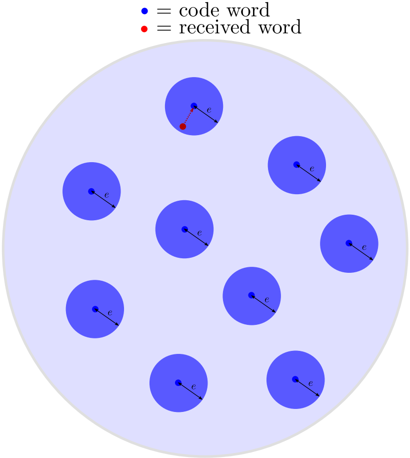
find the polynomial $w(x)$ of degree $\leqslant k$ such that $\#\{i \mid w(\alpha_i) \neq \beta_i\} \leqslant e$

. $(\alpha_1, \ldots, \alpha_n)$ = encoding points
. $(\beta_1, \ldots, \beta_n)$ = received word
. $n - e$ = agreement

well-defined:

. existence of w? by construction 👍

. uniqueness of w? a priori 👎... yet, guaranteed **if** no overlap between the balls of possible received words 👍



unique decoding

decoding:

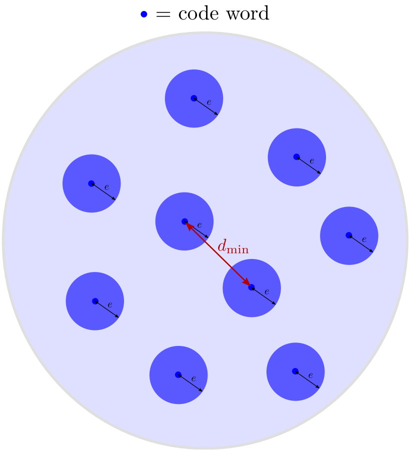
find the polynomial $w(x)$ of degree $\leqslant k$ such that $\#\{i \mid w(\alpha_i) \neq \beta_i\} \leqslant e$

- $(\alpha_1, \ldots, \alpha_n)$ = encoding points
- $(\beta_1, \ldots, \beta_n)$ = received word
- $n - e$ = agreement

well-defined:

- existence of w? by construction
- uniqueness of w? a priori ... yet, guaranteed **if** no overlap between the balls of possible received words

unique decoding bound:
$2e < d_{\text{min}}$



Reed-Solomon case:
$e < \frac{n-k}{2}$

unique decoding

decoding:

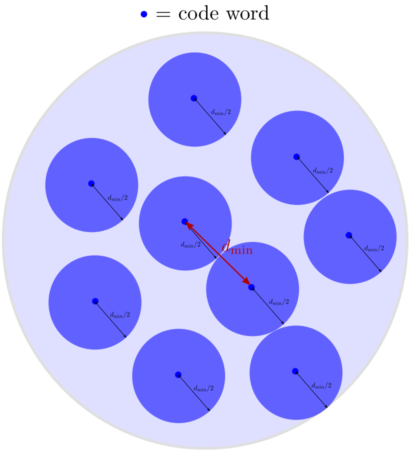
find the polynomial $w(x)$ of degree $\leqslant k$ such that $\#\{i \mid w(\alpha_i) \neq \beta_i\} \leqslant e$

- $(\alpha_1, \ldots, \alpha_n)$ = encoding points
- $(\beta_1, \ldots, \beta_n)$ = received word
- $n - e$ = agreement

well-defined:

- existence of w? by construction 👍
- uniqueness of w? a priori 👎... yet, guaranteed **if** no overlap between the balls of possible received words 👍

unique decoding bound:
$2e < d_{\min}$



Reed-Solomon case:
$e < \frac{n-k}{2}$

bonus: minimum distance for Reed-Solomon codes

- for $v \neq w$ polynomials of degree $\leqslant k$ over the base field $\mathbb{K}$, $(v(\alpha_1), \ldots, v(\alpha_n))$ and $(w(\alpha_1), \ldots, w(\alpha_n))$ agree at $\leqslant k$ positions

  $\Rightarrow$ distance at least $n - k$ between two code words

- for $v = 0$ and $w = (x - \alpha_1) \cdots (x - \alpha_k)$, the code words are $(0, \ldots, 0)$ and $(0, \ldots, 0, w(\alpha_{k+1}), \ldots, w(\alpha_n))$
  $\Rightarrow$ two code words at distance exactly $n - k$

$\Longrightarrow$ minimum distance $d_{\text{min}} = n - k$
(for dimension reasons, this is the best one can hope for)

in this case, unique decoding condition: $e < \dfrac{n-k}{2}$

summary: unique decoding problem

input:

- $\alpha_1, \ldots, \alpha_n$ the n distinct evaluation points in $\mathbb{K}$,
- k the degree bound, e the error-correction radius,
- $(\beta_1, \ldots, \beta_n)$ the received word in $\mathbb{K}^n$

unique decoding requirement: $e < \frac{n-k}{2}$

output: **the** polynomial $w(x)$ in $\mathbb{K}[x]$ such that

$$\deg(w) \leqslant k \qquad \text{and} \qquad \#\{i \mid w(\alpha_i) \neq \beta_i\} \leqslant e$$

summary: unique decoding problem

input:

- $\alpha_1, \ldots, \alpha_n$ the n distinct evaluation points in $\mathbb{K}$,
- k the degree bound, e the error-correction radius,
- $(\beta_1, \ldots, \beta_n)$ the received word in $\mathbb{K}^n$

unique decoding requirement: $e < \frac{n-k}{2}$

output: **the** polynomial $w(x)$ in $\mathbb{K}[x]$ such that

$$\deg(w) \leqslant k \qquad \text{and} \qquad \#\{i \mid w(\alpha_i) \neq \beta_i\} \leqslant e$$

multiple viewpoints + fruitful interactions: **[coding theory]/[computer algebra]**

- linear recurrence generator – Toeplitz linear system – Padé approximation

[Berlekamp'68] [Massey'69] [Brent-Gustavson-Yun'80] [Beckermann-Labahn'94]

- modified extended GCD – rational function reconstruction

[Sugiyama-Kasahara-Hirasawa-Namekawa'75] [Welch-Berlekamp'86]
[Knuth'70] [Schönhage'71] [Moenck'73] [Brent-Gustavson-Yun'80]

- Vandermonde-like linear system – vector rational interpolation

[Olshevsky-Shokrollahi'99] [Kötter-Vardy 2003]
[Morf'74] [Bitmead-Anderson'80] [Pan'90] [van Barel-Bultheel'92] [Beckermann-Labahn'97]

one target complexity: $O(n^3) \to O(n^2) \to O(M(n)\log(n))$

encoding/decoding efficiency: basic remarks

encoding $w(x) \mapsto (w(\alpha_1), \ldots, w(\alpha_n))$

- naive: n times Horner evaluation $O(k)$ $O(nk)$
- fast: $\frac{n}{k}$ times k-point evaluation $O(\frac{n}{k}M(k)\log(k)) \subseteq O(M(n)\log(n))$

points in geometric sequence ⇒ no log factor [Aho-Steiglitz-Ullman'75] [Bostan-Schost 2005]

encoding/decoding efficiency: basic remarks

encoding $w(x) \mapsto (w(\alpha_1), \ldots, w(\alpha_n))$

- naive: n times Horner evaluation $O(k)$ $\qquad O(nk)$
- fast: $\frac{n}{k}$ times k-point evaluation $\qquad O(\frac{n}{k}M(k)\log(k)) \subseteq O(M(n)\log(n))$

points in geometric sequence $\Rightarrow$ no log factor [Aho-Steiglitz-Ullman'75] [Bostan-Schost 2005]

naive decoding

- infinitely lucky decoder: there was no error
  $\rightsquigarrow$ Lagrange interpolation in $O(M(n)\log(n))$ 🥳😎
- very lucky decoder: at most 1 error, unknown position
  $\rightsquigarrow$ trial and error, worst case $O(nM(n)\log(n))$ 🤔😐
- lucky decoder: at most 2 errors, unknown positions
  $\rightsquigarrow$ trial and error, worst case $O(n^2M(n)\log(n))$ 😮😬
- ordinary decoder: at most e errors, unknown positions
  $\rightsquigarrow$ life is tough, complexity exponential in e 😱🌋

next slides = one can be both ordinary and 🥳😎

linear key equations and “rational interpolation” decoding

known interpolant $R(x)$
such that $R(\alpha_i) = \beta_i$

unknown error-locator
$\Lambda(x) = \prod_{i \mid \text{error}} (x - \alpha_i)$

$\Rightarrow \deg(\Lambda) \leqslant e$

key equations: $\Lambda(\alpha_i)R(\alpha_i) = \Lambda(\alpha_i)w(\alpha_i)$ for $1 \leqslant i \leqslant n$

multivariate, non-linear, polynomial system: a priori difficult
(n equations of degree 2 in the $k+1+e$ coefficients of w and Λ)

approach: linearization
introducing the new unknown $\mu = \Lambda w$ of degree $\leqslant k + e$

linear key equations and "rational interpolation" decoding

known interpolant $R(x)$
such that $R(\alpha_i) = \beta_i$

unknown error-locator
$\Lambda(x) = \prod_{i \mid \text{error}} (x - \alpha_i)$

$\Rightarrow \deg(\Lambda) \leqslant e$

key equations: $\Lambda(\alpha_i)R(\alpha_i) = \Lambda(\alpha_i)w(\alpha_i)$ for $1 \leqslant i \leqslant n$

multivariate, non-linear, polynomial system: a priori difficult
(n equations of degree 2 in the $k+1+e$ coefficients of w and Λ)

approach: linearization
introducing the new unknown $\mu = \Lambda w$ of degree $\leqslant k + e$

linear system with n equations and $k+1+2e$ unknowns ($k+1+2e \leqslant n$):

- Gaussian elimination $O(n^3) \to O(n^\omega)$ [Bunch-Hopcroft'74] [Ibarra-Moran-Hui'82]
- $O(n^2) \to O(M(n)\log(n))$ exploiting the Vandermonde-like structure
  [Morf'74] [Bitmead-Anderson'80] [Pan'90] [Olshevsky-Shokrollahi'99]
- $O(n^2) \to O(M(n)\log(n))$ via vector rational interpolation
  [Beckermann'92] [van Barel-Bultheel'92] [Beckermann-Labahn'94,'97] [Kötter-Vardy 2003]

univariate key equation and "rational reconstruction" decoding

known interpolant $R(x)$
such that $R(\alpha_i) = \beta_i$

unknown error-locator
$\Lambda(x) = \prod_{i \mid \text{error}} (x - \alpha_i)$
$\deg(\Lambda) \leqslant e$

unknown linearizer
$\mu(x) = \Lambda(x) w(x)$
$\deg(\mu) \leqslant e + k$

$$\Lambda(\alpha_i) R(\alpha_i) = \mu(\alpha_i) \text{ for } 1 \leqslant i \leqslant n$$

$$\Updownarrow$$

$$\Lambda(x) R(x) = \mu(x) \bmod (x - \alpha_i) \text{ for } 1 \leqslant i \leqslant n$$

$$\Updownarrow$$

[Welch-Berlekamp'86]

$G(x) = \prod_{1 \leqslant i \leqslant n} (x - \alpha_i)$, degree n

univariate key equation: $\Lambda(x) R(x) = \mu(x) \bmod G(x)$

approach: rational reconstruction

$$\begin{cases} \Lambda R = \mu \bmod G \\ \deg(\Lambda) \leqslant e, \quad \deg(\mu) < n - e, \quad \Lambda \text{ monic} \end{cases}$$

note: $e + k < n - e$

univariate key equation and "rational reconstruction" decoding

known interpolant $R(x)$ such that $R(\alpha_i) = \beta_i$

unknown error-locator $\Lambda(x) = \prod_{i \mid \text{error}} (x - \alpha_i)$

$\deg(\Lambda) \leqslant e$

unknown linearizer $\mu(x) = \Lambda(x) w(x)$

$\deg(\mu) \leqslant e + k$

$$\Lambda(\alpha_i) R(\alpha_i) = \mu(\alpha_i) \text{ for } 1 \leqslant i \leqslant n$$

$$\Updownarrow$$

$$\Lambda(x) R(x) = \mu(x) \bmod (x - \alpha_i) \text{ for } 1 \leqslant i \leqslant n$$

$$\Updownarrow$$

[Welch-Berlekamp'86] $G(x) = \prod_{1 \leqslant i \leqslant n} (x - \alpha_i)$, degree n

univariate key equation: $\Lambda(x) R(x) = \mu(x) \bmod G(x)$

approach: rational reconstruction
$$\begin{cases} \Lambda R = \mu \bmod G \\ \deg(\Lambda) \leqslant e, \quad \deg(\mu) < n - e, \quad \Lambda \text{ monic} \end{cases}$$

note: $e + k < n - e$

- unique rational solution $\frac{\mu}{\Lambda}$, which has to be $\frac{\Lambda w}{\Lambda} = w$
- solved by XGCD algorithm stopped at suitable iteration $O(n^2)$
  [Sugiyama-Kasahara-Hirasawa-Namekawa'75] [Modern Computer Algebra, v.z.Gathen-Gerhard, 2003]
- fast XGCD algorithms can be adapted $\rightarrow O(M(n) \log(n))$
  [Knuth'70] [Schönhage'71] [Moenck'73] [Gustavson-Yun'79][Brent-Gustavson-Yun'80]

classical key equation and “Padé approximation” decoding

$$\begin{cases} \Lambda R = \mu \bmod G = \mu + \nu G \quad \text{with } \deg(\Lambda) \leqslant e, \Lambda \text{ monic} \\ \deg(\mu) \leqslant \deg(\Lambda) + k, \quad \deg(\nu) \leqslant \deg(\Lambda) - 1 \end{cases}$$

$\updownarrow$ reverse w.r.t. $x^{n-1+\deg(\Lambda)}$

$$\begin{cases} \bar{\Lambda}\bar{R} = \bar{\mu}x^{n-k-1} + \bar{\nu}\bar{G} = \bar{\nu}\bar{G} \bmod x^{n-k-1} \quad \text{with } \deg(\bar{\Lambda}) \leqslant e, \bar{\Lambda}(0) = 1 \\ \deg(\bar{\mu}) \leqslant \deg(\bar{\Lambda}) + k, \quad \deg(\bar{\nu}) \leqslant \deg(\bar{\Lambda}) - 1 \end{cases}$$

$\downarrow$ $S = \bar{R}/\bar{G} \bmod x^{n-k-1}$ (Newton iteration)

approach: linear recurrence

$$\begin{cases} \bar{\Lambda}S = \bar{\nu} \bmod x^{n-k-1} \\ \deg(\bar{\Lambda}) \leqslant e, \quad \deg(\bar{\nu}) < e, \quad \bar{\Lambda}(0) = 1 \end{cases}$$

classical key equation and “Padé approximation” decoding

$$\begin{cases} \Lambda R = \mu \bmod G = \mu + \nu G & \text{with } \deg(\Lambda) \leqslant e, \Lambda \text{ monic} \\ \deg(\mu) \leqslant \deg(\Lambda) + k, \ \ \deg(\nu) \leqslant \deg(\Lambda) - 1 \end{cases}$$

$\updownarrow$ reverse w.r.t. $x^{n-1+\deg(\Lambda)}$

$$\begin{cases} \bar\Lambda \bar R = \bar\mu x^{n-k-1} + \bar\nu \bar G = \bar\nu \bar G \bmod x^{n-k-1} & \text{with } \deg(\bar\Lambda) \leqslant e, \bar\Lambda(0) = 1 \\ \deg(\bar\mu) \leqslant \deg(\bar\Lambda) + k, \ \ \deg(\bar\nu) \leqslant \deg(\bar\Lambda) - 1 \end{cases}$$

$\downarrow$ $S = \bar R / \bar G \bmod x^{n-k-1}$ (Newton iteration)

approach: linear recurrence

$$\begin{cases} \bar\Lambda S = \bar\nu \bmod x^{n-k-1} \\ \deg(\bar\Lambda) \leqslant e, \ \ \deg(\bar\nu) < e, \ \ \bar\Lambda(0) = 1 \end{cases}$$

- unique rational solution $\bar\nu/\bar\Lambda$, which yields Λ
- coefficients of S: linearly recurrent sequence generated by $\bar\Lambda$

$\rightsquigarrow$ specific algorithms in $O(n^2)$ [Berlekamp'68] [Massey'69]

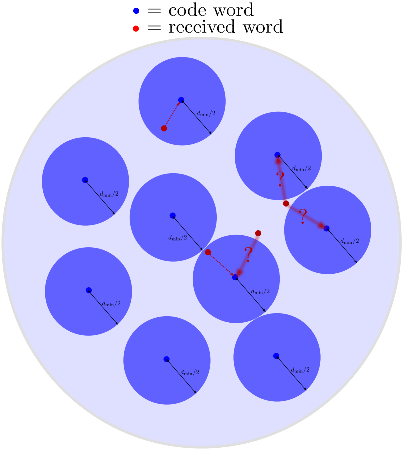
$\rightsquigarrow$ in fact equivalent to the XGCD approach $O(n^2) \to O(M(n)\log(n))$
[Sugiyama et al.'75] [Brent-Gustavson-Yun'80] [Dornstetter'84]

- find $\bar\Lambda$ by homogeneous Toeplitz linear system $O(n^2) \to O(M(n)\log(n))$
- use direct Padé approximation $O(n^2) \to O(M(n)\log(n))$
[Padé 1894] [Sergeyev'86][van Barel-Bultheel'91][Beckermann-Labahn'94]

non-unique decoding

how to decode **more errors**?

- transmission with $\leqslant e$ errors
- where $e \geqslant d_{\min}/2$



non-unique decoding

how to decode **more errors**?

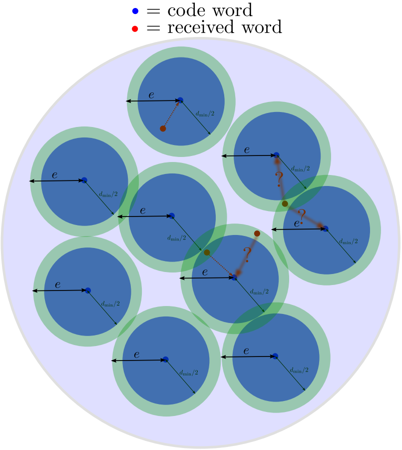
. transmission with $\leqslant e$ errors
. where $e \geqslant d_{\min}/2$

well-defined?

. existence of w: 👍, by construction

. uniqueness of w: 👎, possibly several code words at the same distance

. closest code word not necessarily the sent code word!



non-unique decoding

how to decode **more errors**?

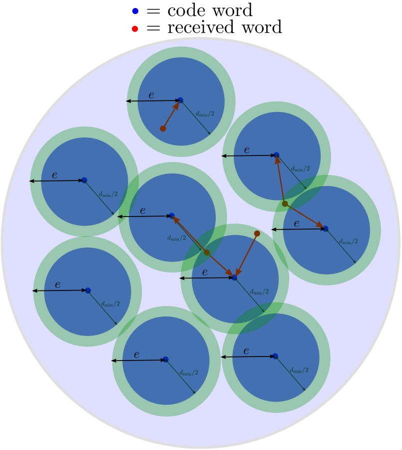
- transmission with $\leqslant e$ errors
- where $e \geqslant d_{\min}/2$

well-defined?

- existence of w: 👍, by construction
- uniqueness of w: 👎, possibly several code words at the same distance
- closest code word not necessarily the sent code word!

list-decoding:
return a **list** of **all** code words at distance $\leqslant e$

[Elias'50s]



list decoding problem

for convenience, we use the agreement parameter $t = n - e$:

$$\#\{i \mid w(\alpha_i) \neq \beta_i\} \leqslant e \quad \Leftrightarrow \quad \#\{i \mid w(\alpha_i) = \beta_i\} \geqslant t$$

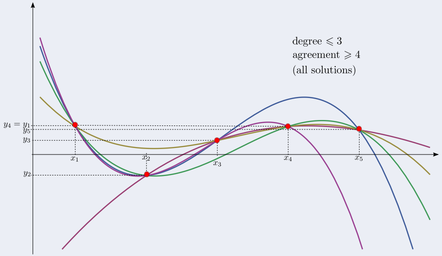
input:

- $\alpha_1, \ldots, \alpha_n$ the n distinct evaluation points in $\mathbb{K}$,
- k the degree bound, $t = n - e$ the agreement,
- $(\beta_1, \ldots, \beta_n)$ the received word in $\mathbb{K}^n$

list decoding requirement: $t^2 > kn$ [Guruswami-Sudan'99]

output: **all** polynomials $w(x)$ in $\mathbb{K}[x]$ such that

$$\deg(w) \leqslant k \qquad \text{and} \qquad \#\{i \mid w(\alpha_i) = \beta_i\} \geqslant t$$



outline

- **computer algebra**
  - efficient algorithms and software
  - for matrices over a field
  - for univariate polynomials
- **Reed-Solomon decoding**
  - context and unique decoding problem
  - key equations and how to solve them
  - correcting more errors?
- **polynomial matrices**
- **efficient list decoding**

outline

- **computer algebra**
  - efficient algorithms and software
  - for matrices over a field
  - for univariate polynomials
- **Reed-Solomon decoding**
  - context and unique decoding problem
  - key equations and how to solve them
  - correcting more errors?
- **polynomial matrices**
  - introduction to vector interpolation
  - core algorithms & shifted normal forms
  - fast divide and conquer interpolation
- **efficient list decoding**

introduction to vector interpolation

⇓ earlier in the talk ⇓

$O(M(d))$

- addition $f+g$, multiplication $f*g$
- division with remainder $f = qg + r$
- truncated inverse $f^{-1} \bmod x^d$

$O(M(d)\log(d))$

- multipoint eval. $f \mapsto f(\alpha_1), \dots, f(\alpha_d)$
- interpolation $f(\alpha_1), \dots, f(\alpha_d) \mapsto f$
- Padé approximation $f = \frac{p}{q} \bmod x^d$
- minpoly of linearly recurrent sequence
- extended GCD $fu + gv = \gcd(f, g)$

⇓ next in the talk ⇓

Padé approximation, sequence minpoly, extended GCD

$O(M(d)\log(d))$ operations in $\mathbb{K}$

matrix versions of these problems

$O(m^\omega M(d)\log(d))$ operations in $\mathbb{K}$

or a tiny bit more for matrix-GCD

Padé approximation:

given power series $f(x)$ at precision d,
given degree constraints $d_1, d_2 > 0$,
$\rightarrow$ compute polynomials $(p(x), q(x))$ of degrees $< (d_1, d_2)$
and such that $f = \frac{p}{q} \bmod x^d$

strong links with linearly recurrent sequences

Padé approximation:

given power series $f(x)$ at precision d,
given degree constraints $d_1, d_2 > 0$,
$\rightarrow$ compute polynomials $(p(x), q(x))$ of degrees $< (d_1, d_2)$
and such that $f = \frac{p}{q} \bmod x^d$

strong links with linearly recurrent sequences

Cauchy interpolation:

given $G(x) = (x - \alpha_1) \cdots (x - \alpha_d) \in \mathbb{K}[x]$,
for pairwise distinct $\alpha_1, \ldots, \alpha_d \in \mathbb{K}$,
given degree constraints $d_1, d_2 > 0$,
$\rightarrow$ compute polynomials $(p(x), q(x))$ of degrees $< (d_1, d_2)$
and such that $f = \frac{p}{q} \bmod G(x)$

Padé approximation:

given power series $f(x)$ at precision d,
given degree constraints $d_1, d_2 > 0$,
$\rightarrow$ compute polynomials $(p(x), q(x))$ of degrees $< (d_1, d_2)$
and such that $f = \frac{p}{q} \bmod x^d$

strong links with linearly recurrent sequences

Cauchy interpolation:

given $G(x) = (x - \alpha_1) \cdots (x - \alpha_d) \in \mathbb{K}[x]$,
for pairwise distinct $\alpha_1, \ldots, \alpha_d \in \mathbb{K}$,
given degree constraints $d_1, d_2 > 0$,
$\rightarrow$ compute polynomials $(p(x), q(x))$ of degrees $< (d_1, d_2)$
and such that $f = \frac{p}{q} \bmod G(x)$

- degree constraints specified by the context
- usual choices have $d_1 + d_2 \approx d$ and existence of a solution

introduction to vector interpolation

approximation and structured linear system

$\mathbb{K} = \mathbb{F}_7$
$f = 2x^7 + 2x^6 + 5x^4 + 2x^2 + 4$
$d = 8, d_1 = 3, d_2 = 6$
$\rightarrow$ look for (p, q) of degree $< (3, 6)$ such that $f = \frac{p}{q} \bmod x^8$

$$\begin{bmatrix} q & p \end{bmatrix} \begin{bmatrix} f \\ -1 \end{bmatrix} = 0 \bmod x^8$$

introduction to vector interpolation

approximation and structured linear system

$\mathbb{K} = \mathbb{F}_7$
$f = 2x^7 + 2x^6 + 5x^4 + 2x^2 + 4$
$d = 8, d_1 = 3, d_2 = 6$
$\rightarrow$ look for (p, q) of degree $< (3, 6)$ such that $f = \frac{p}{q} \bmod x^8$

$$[\ q \quad p\ ] \begin{bmatrix} f \\ -1 \end{bmatrix} = 0 \bmod x^8$$

$$[q_0 \;\; q_1 \;\; q_2 \;\; q_3 \;\; q_4 \;\; 1 \mid p_0 \;\; p_1 \;\; p_2] \left[\begin{array}{cccccccc} 4 & 0 & 2 & 0 & 5 & 0 & 2 & 2 \\ & 4 & 0 & 2 & 0 & 5 & 0 & 2 \\ & & 4 & 0 & 2 & 0 & 5 & 0 \\ & & & 4 & 0 & 2 & 0 & 5 \\ & & & & 4 & 0 & 2 & 0 \\ & & & & & 4 & 0 & 2 \\ \hdashline 6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ & 6 & 0 & 0 & 0 & 0 & 0 & 0 \\ & & 6 & 0 & 0 & 0 & 0 & 0 \end{array}\right] = 0$$

approximation and structured linear system

$\mathbb{K} = \mathbb{F}_7$
$f = 2x^7 + 2x^6 + 5x^4 + 2x^2 + 4$
$d = 8,\ d_1 = 3,\ d_2 = 6$
$\rightarrow$ look for (p, q) of degree $< (3, 6)$ such that $f = \frac{p}{q} \bmod x^8$

$$\begin{bmatrix} q & p \end{bmatrix} \begin{bmatrix} f \\ -1 \end{bmatrix} = 0 \bmod x^8$$

$$\begin{bmatrix} q_0 & q_1 & q_2 & q_3 & q_4 & 1 \mid p_0 & p_1 & p_2 \end{bmatrix} \left[\begin{array}{cccccccc} 4 & 0 & 2 & 0 & 5 & 0 & 2 & 2 \\ & 4 & 0 & 2 & 0 & 5 & 0 & 2 \\ & & 4 & 0 & 2 & 0 & 5 & 0 \\ & & & 4 & 0 & 2 & 0 & 5 \\ & & & & 4 & 0 & 2 & 0 \\ & & & & & 4 & 0 & 2 \\ \hdashline 6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ & 6 & 0 & 0 & 0 & 0 & 0 & 0 \\ & & 6 & 0 & 0 & 0 & 0 & 0 \end{array}\right] = 0$$

Sur la généralisation des fractions continues algébriques;

PAR M. H. PADÉ,

Docteur ès Sciences mathématiques,
Professeur au lycée de Lille.

[**1894**, Journal de mathématiques pures et appliquées]

INTRODUCTION.

M. Hermite s'est, dans un travail récemment paru ([1]), occupé de la généralisation des fractions continues algébriques. La question est de déterminer les polynomes $X_1, X_2, \ldots, X_n$, de degrés $\mu_1, \mu_2, \ldots, \mu_n$, qui satisfont à l'équation

$$S_1 X_1 + S_2 X_2 + \ldots + S_n X_n = S\, x^{\mu_1+\mu_2+\ldots+\mu_n+n-1},$$

$S_1, S_2, \ldots, S_n$ étant des séries entières données, et S une série également entière. Ou plutôt, il s'agit d'obtenir un algorithme qui permette le calcul de proche en proche de ces systèmes de n polynomes, et qui soit analogue à l'algorithme par lequel le numérateur et le dénominateur d'une réduite d'une fraction continue se déduisent des numérateurs et dénominateurs des réduites précédentes. D'élégantes considé-

Hermite-Padé approximation

[Hermite 1893, Padé 1894]

input:

- polynomials $f_1, \ldots, f_m \in \mathbb{K}[x]$
- precision $d \in \mathbb{Z}_{>0}$
- degree bounds $d_1, \ldots, d_m \in \mathbb{Z}_{>0}$

output:

polynomials $p_1, \ldots, p_m \in \mathbb{K}[x]$ such that

- $p_1 f_1 + \cdots + p_m f_m = 0 \bmod x^d$
- $\deg(p_i) < d_i$ for all i

(Padé approximation: particular case $m = 2$ and $f_2 = -1$)

M-Padé approximation / vector rational interpolation
[Cauchy 1821, Mahler 1968]

input:

- polynomials $f_1, \dots, f_m \in \mathbb{K}[x]$
- pairwise distinct points $\alpha_1, \dots, \alpha_d \in \mathbb{K}$
- degree bounds $d_1, \dots, d_m \in \mathbb{Z}_{>0}$

output:

polynomials $p_1, \dots, p_m \in \mathbb{K}[x]$ such that

- $p_1(\alpha_i)f_1(\alpha_i) + \cdots + p_m(\alpha_i)f_m(\alpha_i) = 0$ for all $1 \leqslant i \leqslant d$
- $\deg(p_i) < d_i$ for all i

(rational interpolation: particular case $m = 2$ and $f_2 = -1$)

in this talk: modular equation and fast algebraic algorithms

[van Barel-Bultheel 1992; Beckermann-Labahn 1994, 1997, 2000; Giorgi-Jeannerod-Villard 2003; Storjohann 2006; Zhou-Labahn 2012; Jeannerod-Neiger-Schost-Villard 2017, 2020]

input:

- polynomials $f_1, \ldots, f_m \in \mathbb{K}[x]$
- field elements $\alpha_1, \ldots, \alpha_d \in \mathbb{K}$ $\rightsquigarrow$ not necessarily distinct
- degree bounds $d_1, \ldots, d_m \in \mathbb{Z}_{>0}$ $\rightsquigarrow$ general "shift" $\mathbf{s} \in \mathbb{Z}^m$

output:

polynomials $p_1, \ldots, p_m \in \mathbb{K}[x]$ such that

- $p_1 f_1 + \cdots + p_m f_m = 0 \bmod \prod_{1 \leqslant i \leqslant d}(x - \alpha_i)$
- $\deg(p_i) < d_i$ for all i $\rightsquigarrow$ minimal $\mathbf{s}$-row degree

(Hermite-Padé: $\alpha_1 = \cdots = \alpha_d = 0$; interpolation: pairwise distinct points)

application of vector rational interpolation:
given pairwise distinct points $\{(\alpha_i, \beta_i), 1 \leqslant i \leqslant 8\}$
$= \{(24, 80), (31, 73), (15, 73), (32, 35), (83, 66), (27, 46), (20, 91), (59, 64)\}$,
compute a bivariate polynomial $Q(x, y) \in \mathbb{K}[x, y]$
such that $Q(\alpha_i, \beta_i) = 0$ for $1 \leqslant i \leqslant 8$

$$\left.\begin{array}{l} G(x) = (x-24)\cdots(x-59) \\ R(x) = \text{Lagrange interpolant} \end{array}\right\} \longrightarrow \text{solutions} = \text{ideal } \langle G(x), y - R(x)\rangle$$

solutions of smaller x-degree: $Q(x, y) = Q_0(x) + Q_1(x)y + Q_2(x)y^2$

$$Q(x, R(x)) = \begin{bmatrix} Q_0 & Q_1 & Q_2 \end{bmatrix} \begin{bmatrix} 1 \\ R \\ R^2 \end{bmatrix} = 0 \bmod G(x)$$

- instance of univariate rational vector interpolation
- with a structured input equation (powers of R mod G)

introduction to vector interpolation

interpolation and structured linear system

application of vector rational interpolation:
given pairwise distinct points $\{(\alpha_i, \beta_i), 1 \leqslant i \leqslant 8\}$
$= \{(24, 80), (31, 73), (15, 73), (32, 35), (83, 66), (27, 46), (20, 91), (59, 64)\}$,
compute a bivariate polynomial $Q(x, y) \in \mathbb{K}[x, y]$
such that $Q(\alpha_i, \beta_i) = 0$ for $1 \leqslant i \leqslant 8$

add degree constraints: seek $Q(x, y)$ of the form
$q_{00} + q_{01}x + q_{02}x^2 + q_{03}x^3 + q_{04}x^4 + (q_{10} + q_{11}x + q_{12}x^2)y + q_{20}y^2$:

$$
\left[\begin{array}{ccccc:ccc:c} q_{00} & q_{01} & q_{02} & q_{03} & q_{04} & q_{10} & q_{11} & q_{12} & q_{20} \end{array}\right]
\left[\begin{array}{cccc}
1 & 1 & \cdots & 1 \\
\alpha_1 & \alpha_2 & \cdots & \alpha_8 \\
\alpha_1^2 & \alpha_2^2 & \cdots & \alpha_8^2 \\
\alpha_1^3 & \alpha_2^3 & \cdots & \alpha_8^3 \\
\alpha_1^4 & \alpha_2^4 & \cdots & \alpha_8^4 \\
\hdashline
\beta_1 & \beta_2 & \cdots & \beta_8 \\
\alpha_1\beta_1 & \alpha_2\beta_2 & \cdots & \alpha_8\beta_8 \\
\alpha_1^2\beta_1 & \alpha_2^2\beta_2 & \cdots & \alpha_8^2\beta_8 \\
\hdashline
\beta_1^2 & \beta_2^2 & \cdots & \beta_8^2
\end{array}\right] = 0
$$

- $\mathbb{K}$-linear system
- two levels of structure

$Q(x, y) = (2x^4 + 56x^3 + 42x^2 + 48x + 15) + (72x^2 + 12x + 30)y + y^2$

polynomial matrices enter the arena

why polynomial matrices here?

polynomial matrices enter the arena

why polynomial matrices here?

omitting degree constraints, the set of solutions is
$\mathcal{M} = \{(p_1, \ldots, p_m) \in \mathbb{K}[x]^m \mid p_1 f_1 + \cdots + p_m f_m = 0 \text{ mod } G\}$

recall $G(x) = \prod_{1 \leqslant i \leqslant d}(x - \alpha_i)$

why polynomial matrices here?

omitting degree constraints, the set of solutions is
$\mathcal{M} = \{(p_1, \ldots, p_m) \in \mathbb{K}[x]^m \mid p_1 f_1 + \cdots + p_m f_m = 0 \bmod G\}$

recall $G(x) = \prod_{1 \leqslant i \leqslant d}(x - \alpha_i)$

$\mathcal{M}$ is a "free $\mathbb{K}[x]$-module of rank m", meaning:

- stable under $\mathbb{K}[x]$-linear combinations
- admits a basis consisting of m elements
- basis = $\mathbb{K}[x]$-linear independence + generates all solutions

polynomial matrices enter the arena

why polynomial matrices here?

omitting degree constraints, the set of solutions is
$\mathcal{M} = \{(p_1, \ldots, p_m) \in \mathbb{K}[x]^m \mid p_1 f_1 + \cdots + p_m f_m = 0 \bmod G\}$

recall $G(x) = \prod_{1 \leqslant i \leqslant d}(x - \alpha_i)$

$\mathcal{M}$ is a "free $\mathbb{K}[x]$-module of rank m", meaning:

- stable under $\mathbb{K}[x]$-linear combinations
- admits a basis consisting of m elements
- basis $=$ $\mathbb{K}[x]$-linear independence $+$ generates all solutions

- $\mathcal{M} \subset \mathbb{K}[x]^m \;\Rightarrow\; \mathcal{M}$ has rank $\leqslant m$
- $G(x)\mathbb{K}[x]^m \subset \mathcal{M} \;\Rightarrow\; \mathcal{M}$ has rank $\geqslant m$

remark: solutions are not considered modulo G

e.g. $(G, 0, \ldots, 0)$ is in $\mathcal{M}$ and may appear in a basis

polynomial matrices enter the arena

why polynomial matrices here?

omitting degree constraints, the set of solutions is
$\mathcal{M} = \{(p_1, \ldots, p_m) \in \mathbb{K}[x]^m \mid p_1 f_1 + \cdots + p_m f_m = 0 \text{ mod } G\}$

recall $G(x) = \prod_{1 \leqslant i \leqslant d} (x - \alpha_i)$

basis of solutions:

- square nonsingular matrix $\mathbf{P}$ in $\mathbb{K}[x]^{m \times m}$
- each row of $\mathbf{P}$ is a solution
- any solution is a $\mathbb{K}[x]$-combination $\mathbf{uP}$, $\mathbf{u} \in \mathbb{K}[x]^{1 \times m}$

i.e. $\mathcal{M}$ is the $\mathbb{K}[x]$-row space of $\mathbf{P}$

why polynomial matrices here?

omitting degree constraints, the set of solutions is
$\mathcal{M} = \{(p_1, \ldots, p_m) \in \mathbb{K}[x]^m \mid p_1 f_1 + \cdots + p_m f_m = 0 \bmod G\}$

recall $G(x) = \prod_{1 \leqslant i \leqslant d}(x - \alpha_i)$

basis of solutions:
- square nonsingular matrix $\mathbf{P}$ in $\mathbb{K}[x]^{m \times m}$
- each row of $\mathbf{P}$ is a solution
- any solution is a $\mathbb{K}[x]$-combination $\mathbf{uP}$, $\mathbf{u} \in \mathbb{K}[x]^{1 \times m}$

i.e. $\mathcal{M}$ is the $\mathbb{K}[x]$-row space of $\mathbf{P}$

fact: $\det(\mathbf{P})$ is a divisor of $G(x)$

polynomial matrices enter the arena

why polynomial matrices here?

omitting degree constraints, the set of solutions is
$\mathcal{M} = \{(p_1, \ldots, p_m) \in \mathbb{K}[x]^m \mid p_1 f_1 + \cdots + p_m f_m = 0 \bmod G\}$

recall $G(x) = \prod_{1 \leqslant i \leqslant d}(x - \alpha_i)$

basis of solutions:

- square nonsingular matrix $\mathbf{P}$ in $\mathbb{K}[x]^{m \times m}$
- each row of $\mathbf{P}$ is a solution
- any solution is a $\mathbb{K}[x]$-combination $\mathbf{uP}$, $\mathbf{u} \in \mathbb{K}[x]^{1 \times m}$

i.e. $\mathcal{M}$ is the $\mathbb{K}[x]$-row space of $\mathbf{P}$

fact: $\det(\mathbf{P})$ is a divisor of $G(x)$

fact: any other basis is $\mathbf{UP}$ for $\mathbf{U} \in \mathbb{K}[x]^{m \times m}$ with $\det(\mathbf{U}) \in \mathbb{K} \setminus \{0\}$

polynomial matrices enter the arena

why polynomial matrices here?

omitting degree constraints, the set of solutions is
$\mathcal{M} = \{(p_1, \ldots, p_m) \in \mathbb{K}[x]^m \mid p_1 f_1 + \cdots + p_m f_m = 0 \bmod G\}$

recall $G(x) = \prod_{1 \leqslant i \leqslant d}(x - \alpha_i)$

basis of solutions:

- square nonsingular matrix $\mathbf{P}$ in $\mathbb{K}[x]^{m \times m}$
- each row of $\mathbf{P}$ is a solution
- any solution is a $\mathbb{K}[x]$-combination $\mathbf{uP}$, $\mathbf{u} \in \mathbb{K}[x]^{1 \times m}$

i.e. $\mathcal{M}$ is the $\mathbb{K}[x]$-row space of $\mathbf{P}$

computing a basis of $\mathcal{M}$ with "minimal degrees"

- has many more applications than a single small-degree solution
- is in most cases the fastest known strategy anyway(!)

$\rightsquigarrow$ degree minimality ensured via shifted reduced forms

polynomial matrices: multiplication

$$\mathbf{A} = \begin{bmatrix} 3x+4 & x^3+4x+1 & 4x^2+3 \\ 5 & 5x^2+3x+1 & 5x+3 \\ 3x^3+x^2+5x+3 & 6x+5 & 2x+1 \end{bmatrix} \in \mathbb{K}[x]^{3\times 3}$$

3×3 matrix of degree 3
with entries in $\mathbb{K}[x] = \mathbb{F}_7[x]$

operations on $\mathbb{K}[x]^{m\times m}_{<d}$

- combination of matrix and polynomial computations
- addition in $O(m^2 d)$, naive multiplication in $O(m^3 d^2)$

[Cantor-Kaltofen'91]

multiplication in $O(m^\omega d \log(d) \; + \; m^2 d \log(d) \log\log(d))$

$\in O(m^\omega M(d)) \subset O\tilde{}(m^\omega d)$

2×2 matrices in XGCD, Padé approximation, Berlekamp-Massey, Toeplitz linear systems...

$\rightsquigarrow$ $m \times m$ matrix versions of these problems

- some problems&techniques shared with matrices over $\mathbb{K}$
- some problems&techniques specific to entries in $\mathbb{K}[x]$

polynomial matrices: multiplication

$$\mathbf{A} = \begin{bmatrix} 3x+4 & x^3+4x+1 & 4x^2+3 \\ 5 & 5x^2+3x+1 & 5x+3 \\ 3x^3+x^2+5x+3 & 6x+5 & 2x+1 \end{bmatrix} \in \mathbb{K}[x]^{3\times 3}$$

3×3 matrix of degree 3
with entries in $\mathbb{K}[x] = \mathbb{F}_7[x]$

operations on $\mathbb{K}[x]^{m\times m}_{<d}$

- combination of matrix and polynomial computations
- addition in $O(m^2 d)$, naive multiplication in $O(m^3 d^2)$

[Cantor-Kaltofen'91]

multiplication in $O(m^\omega d \log(d) \;+\; m^2 d \log(d) \log\log(d))$

$\in O(m^\omega \mathsf{M}(d)) \subset O\tilde{~}(m^\omega d)$

applying univariate polynomial techniques directly:

- Newton truncated inversion, matrix-QuoRem $O\tilde{~}(m^\omega d)$ 🥳
- inversion & determinant by evaluation-interpolation $O\tilde{~}(m^{\omega+1} d)$ 😐
- vector rational approximation & interpolation ??? 🤔

applying matrix techniques directly: echelonization is exponential time 🌋

polynomial matrices: main computational problems

reductions of most problems to polynomial matrix multiplication

matrix $m \times m$ of degree d $\rightarrow$ $O\tilde{~}(m^\omega d)$

of "average" degree $\frac{D}{m}$ $\rightarrow$ $O\tilde{~}(m^\omega \frac{D}{m})$

classical matrix operations

- multiplication
- kernel, system solving
- rank, determinant
- inversion $O\tilde{~}(m^3 d)$

univariate specific operations

- truncated inverse, QuoRem
- Hermite-Padé approximation
- vector rational interpolation
- syzygies / modular equations

transformation to **normal forms**

- echelonization: Hermite form
- row reduction: Popov form
- diagonalization: Smith form

polynomial matrices: main computational problems

reductions of most problems to polynomial matrix multiplication

matrix $m \times m$ of degree d $\rightarrow$ $O\tilde{}(m^\omega d)$

of "average" degree $\frac{D}{m}$ $\rightarrow$ $O\tilde{}(m^\omega \frac{D}{m})$

classical matrix operations

- multiplication
- kernel, system solving
- rank, determinant
- inversion $O\tilde{}(m^3 d)$

univariate specific operations

- truncated inverse, QuoRem
- Hermite-Padé approximation
- vector rational interpolation
- syzygies / modular equations

transformation to **normal forms**

- echelonization: Hermite form
- row reduction: Popov form
- diagonalization: Smith form

polynomial matrices: main computational problems

reductions of most problems to polynomial matrix multiplication

matrix $m \times m$ of degree d $\rightarrow$ $O\tilde{}(m^\omega d)$

of "average" degree $\frac{D}{m}$ $\rightarrow$ $O\tilde{}(m^\omega \frac{D}{m})$

classical matrix operations

- multiplication
- kernel, system solving
- rank, determinant
- inversion $O\tilde{}(m^3 d)$

univariate specific operations

- truncated inverse, QuoRem
- Hermite-Padé approximation
- vector rational interpolation
- syzygies / modular equations

transformation to **normal forms**

- echelonization: Hermite form
- row reduction: Popov form
- diagonalization: Smith form

polynomial matrices: main computational problems

reductions of most problems to polynomial matrix multiplication

matrix $m \times m$ of degree d $\rightarrow$ $O\tilde{}(m^\omega d)$

of "average" degree $\frac{D}{m}$ $\rightarrow$ $O\tilde{}(m^\omega \frac{D}{m})$

classical matrix operations

- multiplication
- kernel, system solving
- rank, determinant
- inversion $O\tilde{}(m^3 d)$

univariate specific operations

- truncated inverse, QuoRem
- Hermite-Padé approximation
- vector rational interpolation
- syzygies / modular equations

transformation to **normal forms**

- echelonization: Hermite form
- row reduction: Popov form
- diagonalization: Smith form

Hermite and Popov forms

working over $\mathbb{K} = \mathbb{Z}/7\mathbb{Z}$

$$\mathbf{A} = \begin{bmatrix} 3x+4 & x^3+4x+1 & 4x^2+3 \\ 5 & 5x^2+3x+1 & 5x+3 \\ 3x^3+x^2+5x+3 & 6x+5 & 2x+1 \end{bmatrix}$$

using elementary row operations, transform $\mathbf{A}$ into...

Hermite form $\mathbf{H} = \begin{bmatrix} x^6+6x^4+x^3+x+4 & 0 & 0 \\ 5x^5+5x^4+6x^3+2x^2+6x+3 & x & 0 \\ 3x^4+5x^3+4x^2+6x+1 & 5 & 1 \end{bmatrix}$

Popov form $\mathbf{P} = \begin{bmatrix} x^3+5x^2+4x+1 & 2x+4 & 3x+5 \\ 1 & x^2+2x+3 & x+2 \\ 3x+2 & 4x & x^2 \end{bmatrix}$

Hermite and Popov forms

nonsingular $\mathbf{A} \in \mathbb{K}[x]^{m\times m}$

elementary row transformations

Hermite form [Hermite, 1851]
- triangular
- column normalized

$$\begin{bmatrix} \mathbf{16} & & & \\ 15 & \mathbf{0} & & \\ 15 & & \mathbf{0} & \\ 15 & & & \mathbf{0} \end{bmatrix} \qquad \begin{bmatrix} \mathbf{4} & & & \\ 3 & \mathbf{7} & & \\ 1 & 5 & \mathbf{3} & \\ 3 & 6 & 1 & \mathbf{2} \end{bmatrix}$$

Hermite and Popov forms

nonsingular $\mathbf{A} \in \mathbb{K}[x]^{m \times m}$

elementary row transformations

Hermite form [Hermite, 1851]
- triangular
- column normalized

$$\begin{bmatrix} \mathbf{16} & & & \\ 15 & \mathbf{0} & & \\ 15 & & \mathbf{0} & \\ 15 & & & \mathbf{0} \end{bmatrix} \quad \begin{bmatrix} \mathbf{4} & & & \\ 3 & \mathbf{7} & & \\ 1 & 5 & \mathbf{3} & \\ 3 & 6 & 1 & \mathbf{2} \end{bmatrix}$$

Popov form [Popov, 1972]
- minimal row degrees
- column normalized

$$\begin{bmatrix} \mathbf{4} & 3 & 3 & 3 \\ 3 & \mathbf{4} & 3 & 3 \\ 3 & 3 & \mathbf{4} & 3 \\ 3 & 3 & 3 & \mathbf{4} \end{bmatrix} \quad \begin{bmatrix} \mathbf{7} & 0 & 1 & 5 \\ 0 & \mathbf{1} & & 0 \\ & & \mathbf{2} & \\ 6 & 0 & 1 & \mathbf{6} \end{bmatrix}$$

Hermite and Popov forms

nonsingular $\mathbf{A} \in \mathbb{K}[x]^{m \times m}$

elementary row transformations

Hermite form [Hermite, 1851]
- triangular
- column normalized

Popov form [Popov, 1972]
- minimal row degrees
- column normalized

$$\begin{bmatrix} \mathbf{16} & & & \\ 15 & \mathbf{0} & & \\ 15 & & \mathbf{0} & \\ 15 & & & \mathbf{0} \end{bmatrix} \quad \begin{bmatrix} \mathbf{4} & & & \\ 3 & \mathbf{7} & & \\ 1 & 5 & \mathbf{3} & \\ 3 & 6 & 1 & \mathbf{2} \end{bmatrix} \qquad \begin{bmatrix} \mathbf{4} & 3 & 3 & 3 \\ 3 & \mathbf{4} & 3 & 3 \\ 3 & 3 & \mathbf{4} & 3 \\ 3 & 3 & 3 & \mathbf{4} \end{bmatrix} \quad \begin{bmatrix} \mathbf{7} & 0 & 1 & 5 \\ 0 & \mathbf{1} & & 0 \\ & & \mathbf{2} & \\ 6 & 0 & 1 & \mathbf{6} \end{bmatrix}$$

$\preccurlyeq_{\mathrm{pot}}$ reduced Gröbner basis $\preccurlyeq_{\mathrm{top}}$

$\mathbb{K}[x]$-module $\mathcal{M} \subset \mathbb{K}[x]^{1 \times m}$ of rank m

Hermite and Popov forms

nonsingular $\mathbf{A} \in \mathbb{K}[x]^{m \times m}$

elementary row transformations

Hermite form [Hermite, 1851]
- triangular
- column normalized

Popov form [Popov, 1972]
- minimal row degrees
- column normalized

$$\begin{bmatrix} \mathbf{16} & & & \\ 15 & \mathbf{0} & & \\ 15 & & \mathbf{0} & \\ 15 & & & \mathbf{0} \end{bmatrix} \quad \begin{bmatrix} \mathbf{4} & & & \\ 3 & \mathbf{7} & & \\ 1 & 5 & \mathbf{3} & \\ 3 & 6 & 1 & \mathbf{2} \end{bmatrix} \qquad \begin{bmatrix} \mathbf{4} & 3 & 3 & 3 \\ 3 & \mathbf{4} & 3 & 3 \\ 3 & 3 & \mathbf{4} & 3 \\ 3 & 3 & 3 & \mathbf{4} \end{bmatrix} \quad \begin{bmatrix} \mathbf{7} & 0 & 1 & 5 \\ 0 & \mathbf{1} & & 0 \\ & & \mathbf{2} & \\ 6 & 0 & 1 & \mathbf{6} \end{bmatrix}$$

invariant: $D = \deg(\det(\mathbf{A})) = 4+7+3+2 = 7+1+2+6$

- average column degree is $\frac{D}{m}$
- size of object is $mD + m^2 = m^2(\frac{D}{m}+1)$

target cost: $O\tilde{~}(m^{\omega}\frac{D}{m})$

Hermite and Popov forms

nonsingular $\mathbf{A} \in \mathbb{K}[x]^{m \times m}$

elementary row transformations

Hermite form [Hermite, 1851]
- triangular
- column normalized

Popov form [Popov, 1972]
- minimal row degrees
- column normalized

$$\begin{bmatrix} \mathbf{16} & & & \\ 15 & \mathbf{0} & & \\ 15 & & \mathbf{0} & \\ 15 & & & \mathbf{0} \end{bmatrix} \quad \begin{bmatrix} \mathbf{4} & & & \\ 3 & \mathbf{7} & & \\ 1 & 5 & \mathbf{3} & \\ 3 & 6 & 1 & \mathbf{2} \end{bmatrix}$$

$$\begin{bmatrix} \mathbf{4} & 3 & 3 & 3 \\ 3 & \mathbf{4} & 3 & 3 \\ 3 & 3 & \mathbf{4} & 3 \\ 3 & 3 & 3 & \mathbf{4} \end{bmatrix} \quad \begin{bmatrix} \mathbf{7} & 0 & 1 & 5 \\ 0 & \mathbf{1} & & 0 \\ & & \mathbf{2} & \\ 6 & 0 & 1 & \mathbf{6} \end{bmatrix}$$

[Beckermann-Labahn-Villard, 1999; Mulders-Storjohann, 2003]

shifted reduced form:
arbitrary degree constraints + **no** column normalization

$\approx$ minimal, non-reduced, $\prec$-Gröbner basis

shifted forms

shift: integer tuple $\mathbf{s} = (s_1, \ldots, s_m)$ acting as column weights

$\rightarrow$ connects Popov and Hermite forms

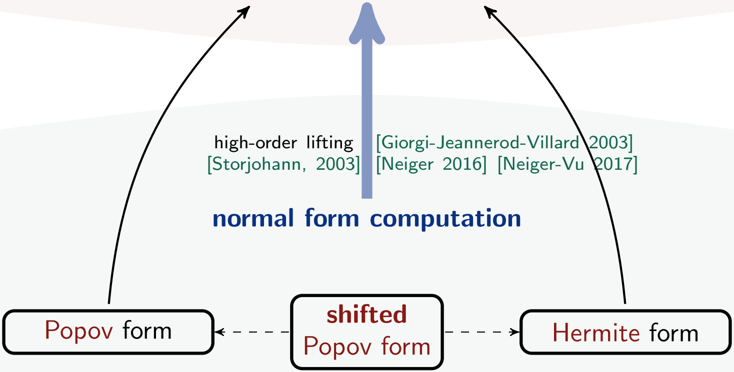
$\mathbf{s} = (0,0,0,0)$ Popov	$\begin{bmatrix} \mathbf{4} & 3 & 3 & 3 \\ 3 & \mathbf{4} & 3 & 3 \\ 3 & 3 & \mathbf{4} & 3 \\ 3 & 3 & 3 & \mathbf{4} \end{bmatrix}$	$\begin{bmatrix} \mathbf{7} & 0 & 1 & 5 \\ 0 & \mathbf{1} & & 0 \\ & & \mathbf{2} & \\ 6 & 0 & 1 & \mathbf{6} \end{bmatrix}$
$\mathbf{s} = (0,2,4,6)$ $\mathbf{s}$-Popov	$\begin{bmatrix} \mathbf{7} & 4 & 2 & 0 \\ 6 & \mathbf{5} & 2 & 0 \\ 6 & 4 & \mathbf{3} & 0 \\ 6 & 4 & 2 & \mathbf{1} \end{bmatrix}$	$\begin{bmatrix} \mathbf{8} & 5 & 1 & \\ 7 & \mathbf{6} & 1 & \\ & & \mathbf{2} & \\ 0 & 1 & & \mathbf{0} \end{bmatrix}$
$\mathbf{s} = (0, D, 2D, 3D)$ Hermite	$\begin{bmatrix} \mathbf{16} & & & \\ 15 & \mathbf{0} & & \\ 15 & & \mathbf{0} & \\ 15 & & & \mathbf{0} \end{bmatrix}$	$\begin{bmatrix} \mathbf{4} & & & \\ 3 & \mathbf{7} & & \\ 1 & 5 & \mathbf{3} & \\ 3 & 6 & 1 & \mathbf{2} \end{bmatrix}$

- normal form, average column degree D/m
- shifts arise naturally in algorithms (approximants, kernel, ...)
- they allow one to specify non-uniform degree constraints

from normal forms to relations

$$\begin{cases} p_1 f_{11} + \cdots + p_m f_{1m} & = & 0 \bmod g_1 \\ \vdots & \vdots & \vdots \\ p_1 f_{n1} + \cdots + p_m f_{nm} & = & 0 \bmod g_n \end{cases}$$

reconstruction as relations



software development for polynomial matrices

```
sage: M.degree_matrix(shifts=[-1,2], row_wise=False)
[ 0 -2 -1]
[ 5 -2 -2]
```

hermite_form(*include_zero_rows=True, transformation=False*)

Return the Hermite form of this matrix.

The Hermite form is also normalized, i.e., the pivot polynomials are monic.

INPUT:

- `include_zero_rows` – boolean (default: `True`); if `False`, the zero rows in the output deleted
- `transformation` – boolean (default: `False`); if `True`, return the transformation mat

OUTPUT:

```
sage: M.<x> = GF(7)[]
sage: A = matrix(M, 2, 3, [x, 1, 2*x, x, 1+x, 2])
sage: A.hermite_form()
[     x       1     2*x]
[     0       x 5*x + 2]
sage: A.hermite_form(transformation=True)
(
[     x       1     2*x]  [1 0]
[     0       x 5*x + 2], [6 1]
)
sage: A = matrix(M, 2, 3, [x, 1, 2*x, 2*x, 2, 4*x])
sage: A.hermite_form(transformation=True, include_zero_rows=False)
([  x   1 2*x], [0 4])
sage: H, U = A.hermite_form(transformation=True, include_zero_rows=True); H, U
(
[  x   1 2*x]  [0 4]
[  0   0   0], [5 1]
)
sage: U * A == H
True
sage: H, U = A.hermite_form(transformation=True, include_zero_rows=False)
sage: U * A
[  x   1 2*x]
sage: U * A == H
True
```

See also: `is_hermite()` .

is_hermite(*row_wise=True, lower_echelon=False, include_zero_vectors=True*)

Return a boolean indicating whether this matrix is in Hermite form.

```
    // order that remains to be dealt with
    VecLong rem_order(order);

    // indices of columns/orders that remain to be dealt with
    VecLong rem_index(cdim);
    std::iota(rem_index.begin(), rem_index.end(), 0);

    // all along the algorithm, shift = shifted row degrees of approximant ba
    // (initially, input shift = shifted row degree of the identity matrix)

    while (not rem_order.empty())
    {
        /** Invariant:
         *  - appbas is a shift-ordered weak Popov approximant basis for
         *  (pmat,reached_order) where doneorder is the tuple such that
         *  -->reached_order[j] + rem_order[j] == order[j] for j appearing in
         *  -->reached_order[j] == order[j] for j not appearing in rem_index
         *  - shift == the "input shift"-row degree of appbas
         *  - residual == submatrix of columns (appbas * pmat)[:,j] for all j
[j] < order[j]]

            j = std::distance(rem_order.begin(), std::max_element(rem_order.be
);

        long deg = order[rem_index[j]] - rem_order[j];

        // record the coefficients of degree deg of the column j of residual
        // also keep track of which of these are nonzero,
        // and among the nonzero ones, which is the first with smallest shift
        Vec<zz_p> const_residual;
        const_residual.SetLength(rdim);
        VecLong indices_nonzero;
        long piv = -1;
        for (long i = 0; i < rdim; ++i)
        {
            const_residual[i] = coeff(residual[i][j],deg);
            if (const_residual[i] != 0)
            {
                indices_nonzero.push_back(i);
                if (piv<0 || shift[i] < shift[piv])
                    piv = i;
            }
        }

        // if indices_nonzero is empty, const_residual is already zero, there
        if (not indices_nonzero.empty())
        {
            // update all rows of appbas and residual in indices_nonzero exce
src/mat_lzz_pX_approximant.cpp
```

open-source mathematics software system

Python/Cython

goals: **complete, robust, available**

(more than 60k downloads per month)

software development for polynomial matrices

open-source mathematics software system

sage — *Python/Cython*

goals: **complete, robust, available**

(more than 60k downloads per month)

high-performance exact linear algebra

LinBox – fflas-ffpack — *C/C++*

goal: **optimized basic operations**

memory cost, vectorization, multithreading

software development for polynomial matrices

```
sage: M.<x> = GF(7)[]
sage: A = matrix(M, 2, 3, [x, 1, 2*x, x, 1+x, 2])
sage: A.hermite_form()
[      x       1     2*x]
[      0       x 5*x + 2]
sage: A.hermite_form(transformation=True)
(
[      x       1     2*x]  [1 0]
[      0       x 5*x + 2], [6 1]
)
sage: A = matrix(M, 2, 3, [x, 1, 2*x, 2*x, 2, 4*x])
sage: A.hermite_form(transformation=True, include_zero_rows=False)
([  x   1 2*x], [0 4])
sage: H, U = A.hermite_form(transformation=True, include_zero_rows=True); H, U
(
[  x   1 2*x]  [0 4]
[  0   0   0], [5 1]
)
sage: U * A == H
True
sage: H, U = A.hermite_form(transformation=True, include_zero_rows=False)
sage: U * A
[  x   1 2*x]
sage: U * A == H
True
```

See also: is_hermite() .

is_hermite(*row_wise=True, lower_echelon=False, include_zero_vectors=True*)

Return a boolean indicating whether this matrix is in Hermite form.

```
            j = std::distance(rem_order.begin(), std::max_element(rem_order.be
);

        long deg = order[rem_index[j]] - rem_order[j];

        // record the coefficients of degree deg of the column j of residual
        // also keep track of which of these are nonzero,
        // and among the nonzero ones, which is the first with smallest shift
        Vec<zz_p> const_residual;
        const_residual.SetLength(rdim);
        VecLong indices_nonzero;
        long piv = -1;
        for (long i = 0; i < rdim; ++i)
        {
            const_residual[i] = coeff(residual[i][j],deg);
            if (const_residual[i] != 0)
            {
                indices_nonzero.push_back(i);
                if (piv<0 || shift[i] < shift[piv])
                    piv = i;
            }
        }

        // if indices_nonzero is empty, const_residual is already zero, there
        if (not indices_nonzero.empty())
        {
            // update all rows of appbas and residual in indices_nonzero exce
```

src/mat_lzz_pX_approximant.cpp

open-source mathematics software system

sage *Python/Cython*

goals: **complete, robust, available**

(more than 60k downloads per month)

high-performance exact linear algebra

LinBox – fflas-ffpack *C/C++*

goal: **optimized basic operations**

memory cost, vectorization, multithreading

software development for polynomial matrices

Polynomial Matrix Library *C/C++*

403 files, 59k lines of code, including 17k lines of comments

`https://github.com/vneiger/pml`

[Hyun-Neiger-Schost '19]

- current version based on NTL
- work-in-progress version based on FLINT
- **welcome** comments, suggestions, contributions

"hey, this doesn't work!"
"yo, plans for implementing this?"
"how to decode RS codes with PML?"

wide range of algorithms

efficiency = state of the art

kernel, high-order lifting,
system solving, reduced form. . .

polynomial matrices: two open questions

deterministic Smith form

$$\begin{bmatrix} & & \\ & \mathbf{A} & \\ & & \end{bmatrix} \longrightarrow \begin{bmatrix} s_1 & & & \\ & s_2 & & \\ & & \ddots & \\ & & & s_m \end{bmatrix}$$

s_{i+1} divides s_i

- complexity $O\tilde{}(m^\omega \frac{D}{m})$ [Storjohann'03]
- Las Vegas randomized algorithm
- requires large field $\mathbb{K}$

deterministic algo in $O\tilde{}(m^\omega \frac{D}{m})$?

polynomial matrices: two open questions

deterministic Smith form

$$\begin{bmatrix} & & \\ & \mathbf{A} & \\ & & \end{bmatrix} \longrightarrow \begin{bmatrix} s_1 & & & \\ & s_2 & & \\ & & \ddots & \\ & & & s_m \end{bmatrix}$$

s_{i+1} divides s_i

- complexity $O\tilde{~}(m^\omega \frac{D}{m})$ [Storjohann'03]
- Las Vegas randomized algorithm
- requires large field $\mathbb{K}$

deterministic algo in $O\tilde{~}(m^\omega \frac{D}{m})$?

algebraic interpolants

= main step of Sudan decoding

$$p_1 f_1 + p_2 f_2 + \cdots + p_m f_m = 0 \bmod G$$

$\downarrow$ structured f_i's

$$p_1 1 + p_2 R + \cdots + p_m R^{m-1} = 0 \bmod G$$

- most algorithms ignore the structure
- recent progress [Villard'18]
- restrictive: genericity, specific m & d

how to leverage this structure?

fast divide and conquer interpolation

iterative algorithm [van Barel-Bultheel / Beckermann-Labahn / Kötter-Vardy]

parameters: $d = 8$ $m = 4$ $\mathbf{s} = (0, 2, 4, 6)$, base field $\mathbb{F}_{97}$

input: $(24, 31, 15, 32, 83, 27, 20, 59)$ and $\mathbf{F} = [1 \;\; R \;\; R^2 \;\; R^3]^\mathsf{T}$

iteration: $i = 1$ **point:** 24, 31, 15, 32, 83, 27, 20, 59

shift $[0 \quad 2 \quad 4 \quad 6]$

basis $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

values $\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 80 & 73 & 73 & 35 & 66 & 46 & 91 & 64 \\ 95 & 91 & 91 & 61 & 88 & 79 & 36 & 22 \\ 34 & 47 & 47 & 1 & 85 & 45 & 75 & 50 \end{bmatrix}$

fast divide and conquer interpolation

iterative algorithm [van Barel-Bultheel / Beckermann-Labahn / Kötter-Vardy]

parameters: $d = 8$ $m = 4$ $\mathbf{s} = (0, 2, 4, 6)$, base field $\mathbb{F}_{97}$

input: $(24, 31, 15, 32, 83, 27, 20, 59)$ and $\mathbf{F} = [1 \;\; R \;\; R^2 \;\; R^3]^T$

iteration: $i = 1$ **point:** 24, 31, 15, 32, 83, 27, 20, 59

shift $[0 \quad 2 \quad 4 \quad 6]$

basis
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

values
$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 80 & 73 & 73 & 35 & 66 & 46 & 91 & 64 \\ 95 & 91 & 91 & 61 & 88 & 79 & 36 & 22 \\ 34 & 47 & 47 & 1 & 85 & 45 & 75 & 50 \end{bmatrix}$$

fast divide and conquer interpolation

iterative algorithm [van Barel-Bultheel / Beckermann-Labahn / Kötter-Vardy]

parameters: $d = 8$ $m = 4$ $\mathbf{s} = (0, 2, 4, 6)$, base field $\mathbb{F}_{97}$

input: $(24, 31, 15, 32, 83, 27, 20, 59)$ and $\mathbf{F} = [1 \;\; R \;\; R^2 \;\; R^3]^T$

iteration: $i = 1$ **point:** 24, 31, 15, 32, 83, 27, 20, 59

shift $[0 \;\; 2 \;\; 4 \;\; 6]$

basis

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 17 & 1 & 0 & 0 \\ 2 & 0 & 1 & 0 \\ 63 & 0 & 0 & 1 \end{bmatrix}$$

values

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 90 & 90 & 52 & 83 & 63 & 11 & 81 \\ 0 & 93 & 93 & 63 & 90 & 81 & 38 & 24 \\ 0 & 13 & 13 & 64 & 51 & 11 & 41 & 16 \end{bmatrix}$$

fast divide and conquer interpolation

iterative algorithm [van Barel-Bultheel / Beckermann-Labahn / Kötter-Vardy]

parameters: $d = 8$ $m = 4$ $\mathbf{s} = (0, 2, 4, 6)$, base field $\mathbb{F}_{97}$

input: $(24, 31, 15, 32, 83, 27, 20, 59)$ and $\mathbf{F} = [1 \;\; R \;\; R^2 \;\; R^3]^T$

iteration: $i = 1$ **point:** 24, 31, 15, 32, 83, 27, 20, 59

shift $[1 \;\; 2 \;\; 4 \;\; 6]$

basis

$$\begin{bmatrix} x + 73 & 0 & 0 & 0 \\ 17 & 1 & 0 & 0 \\ 2 & 0 & 1 & 0 \\ 63 & 0 & 0 & 1 \end{bmatrix}$$

values

$$\begin{bmatrix} 0 & 7 & 88 & 8 & 59 & 3 & 93 & 35 \\ 0 & 90 & 90 & 52 & 83 & 63 & 11 & 81 \\ 0 & 93 & 93 & 63 & 90 & 81 & 38 & 24 \\ 0 & 13 & 13 & 64 & 51 & 11 & 41 & 16 \end{bmatrix}$$

fast divide and conquer interpolation

iterative algorithm [van Barel-Bultheel / Beckermann-Labahn / Kötter-Vardy]

parameters: $d = 8$ $m = 4$ $\mathbf{s} = (0, 2, 4, 6)$, base field $\mathbb{F}_{97}$

input: $(24, 31, 15, 32, 83, 27, 20, 59)$ and $\mathbf{F} = [1 \;\; R \;\; R^2 \;\; R^3]^T$

iteration: $i = 2$ **point:** 24, 31, 15, 32, 83, 27, 20, 59

shift $[1 \;\; 2 \;\; 4 \;\; 6]$

basis
$$\begin{bmatrix} x + 73 & 0 & 0 & 0 \\ 17 & 1 & 0 & 0 \\ 2 & 0 & 1 & 0 \\ 63 & 0 & 0 & 1 \end{bmatrix}$$

values
$$\begin{bmatrix} 0 & 7 & 88 & 8 & 59 & 3 & 93 & 35 \\ 0 & 90 & 90 & 52 & 83 & 63 & 11 & 81 \\ 0 & 93 & 93 & 63 & 90 & 81 & 38 & 24 \\ 0 & 13 & 13 & 64 & 51 & 11 & 41 & 16 \end{bmatrix}$$

fast divide and conquer interpolation

iterative algorithm [van Barel-Bultheel / Beckermann-Labahn / Kötter-Vardy]

parameters: $d = 8$ $\quad m = 4$ $\quad \mathbf{s} = (0, 2, 4, 6)$, base field $\mathbb{F}_{97}$

input: $(24, 31, 15, 32, 83, 27, 20, 59)$ and $\mathbf{F} = [1 \;\; R \;\; R^2 \;\; R^3]^\mathsf{T}$

iteration: $i = 2$ **point:** 24, 31, 15, 32, 83, 27, 20, 59

shift $[1 \quad 2 \quad 4 \quad 6]$

basis
$$\begin{bmatrix} x + 73 & 0 & 0 & 0 \\ x + 90 & 1 & 0 & 0 \\ 56x + 16 & 0 & 1 & 0 \\ 12x + 66 & 0 & 0 & 1 \end{bmatrix}$$

values
$$\begin{bmatrix} 0 & 7 & 88 & 8 & 59 & 3 & 93 & 35 \\ 0 & 0 & 81 & 60 & 45 & 66 & 7 & 19 \\ 0 & 0 & 74 & 26 & 96 & 55 & 8 & 44 \\ 0 & 0 & 2 & 63 & 80 & 47 & 90 & 48 \end{bmatrix}$$

fast divide and conquer interpolation

iterative algorithm [van Barel-Bultheel / Beckermann-Labahn / Kötter-Vardy]

parameters: $d = 8$ $m = 4$ $\mathbf{s} = (0, 2, 4, 6)$, base field $\mathbb{F}_{97}$

input: $(24, 31, 15, 32, 83, 27, 20, 59)$ and $\mathbf{F} = [1 \;\; R \;\; R^2 \;\; R^3]^T$

iteration: $i = 2$ **point:** 24, 31, 15, 32, 83, 27, 20, 59

shift $[2 \;\; 2 \;\; 4 \;\; 6]$

basis

$$\begin{bmatrix} x^2 + 42x + 65 & 0 & 0 & 0 \\ x + 90 & 1 & 0 & 0 \\ 56x + 16 & 0 & 1 & 0 \\ 12x + 66 & 0 & 0 & 1 \end{bmatrix}$$

values

$$\begin{bmatrix} 0 & 0 & 47 & 8 & 61 & 85 & 44 & 10 \\ 0 & 0 & 81 & 60 & 45 & 66 & 7 & 19 \\ 0 & 0 & 74 & 26 & 96 & 55 & 8 & 44 \\ 0 & 0 & 2 & 63 & 80 & 47 & 90 & 48 \end{bmatrix}$$

fast divide and conquer interpolation

iterative algorithm [van Barel-Bultheel / Beckermann-Labahn / Kötter-Vardy]

parameters: $d = 8$ $m = 4$ $\mathbf{s} = (0, 2, 4, 6)$, base field $\mathbb{F}_{97}$

input: $(24, 31, 15, 32, 83, 27, 20, 59)$ and $\mathbf{F} = [1 \;\; R \;\; R^2 \;\; R^3]^T$

iteration: $i = 3$ **point:** 24, 31, 15, 32, 83, 27, 20, 59

shift $[2 \quad 2 \quad 4 \quad 6]$

basis

$$\begin{bmatrix} x^2 + 42x + 65 & 0 & 0 & 0 \\ x + 90 & 1 & 0 & 0 \\ 56x + 16 & 0 & 1 & 0 \\ 12x + 66 & 0 & 0 & 1 \end{bmatrix}$$

values

$$\begin{bmatrix} 0 & 0 & 47 & 8 & 61 & 85 & 44 & 10 \\ 0 & 0 & 81 & 60 & 45 & 66 & 7 & 19 \\ 0 & 0 & 74 & 26 & 96 & 55 & 8 & 44 \\ 0 & 0 & 2 & 63 & 80 & 47 & 90 & 48 \end{bmatrix}$$

fast divide and conquer interpolation

iterative algorithm [van Barel-Bultheel / Beckermann-Labahn / Kötter-Vardy]

parameters: $d = 8$ $m = 4$ $\mathbf{s} = (0, 2, 4, 6)$, base field $\mathbb{F}_{97}$

input: $(24, 31, 15, 32, 83, 27, 20, 59)$ and $\mathbf{F} = [1 \;\; R \;\; R^2 \;\; R^3]^T$

iteration: $i = 3$ **point:** 24, 31, 15, 32, 83, 27, 20, 59

shift $[3 \quad 2 \quad 4 \quad 6]$

basis
$$\begin{bmatrix} x^3 + 27x^2 + 17x + 92 & 0 & 0 & 0 \\ 54x^2 + 38x + 11 & 1 & 0 & 0 \\ 17x^2 + 91x + 54 & 0 & 1 & 0 \\ 66x^2 + 68x + 88 & 0 & 0 & 1 \end{bmatrix}$$

values
$$\begin{bmatrix} 0 & 0 & 0 & 39 & 74 & 50 & 26 & 52 \\ 0 & 0 & 0 & 7 & 41 & 0 & 55 & 74 \\ 0 & 0 & 0 & 65 & 66 & 45 & 77 & 20 \\ 0 & 0 & 0 & 9 & 32 & 31 & 84 & 29 \end{bmatrix}$$

fast divide and conquer interpolation

iterative algorithm [van Barel-Bultheel / Beckermann-Labahn / Kötter-Vardy]

parameters: $d = 8$ $m = 4$ $\mathbf{s} = (0, 2, 4, 6)$, base field $\mathbb{F}_{97}$

input: $(24, 31, 15, 32, 83, 27, 20, 59)$ and $\mathbf{F} = [1 \;\; R \;\; R^2 \;\; R^3]^T$

iteration: $i = 4$ **point:** 24, 31, 15, 32, 83, 27, 20, 59

shift $[3 \quad 2 \quad 4 \quad 6]$

basis

$$\begin{bmatrix} x^3 + 27x^2 + 17x + 92 & 0 & 0 & 0 \\ 54x^2 + 38x + 11 & 1 & 0 & 0 \\ 17x^2 + 91x + 54 & 0 & 1 & 0 \\ 66x^2 + 68x + 88 & 0 & 0 & 1 \end{bmatrix}$$

values

$$\begin{bmatrix} 0 & 0 & 0 & 39 & 74 & 50 & 26 & 52 \\ 0 & 0 & 0 & 7 & 41 & 0 & 55 & 74 \\ 0 & 0 & 0 & 65 & 66 & 45 & 77 & 20 \\ 0 & 0 & 0 & 9 & 32 & 31 & 84 & 29 \end{bmatrix}$$

fast divide and conquer interpolation

iterative algorithm [van Barel-Bultheel / Beckermann-Labahn / Kötter-Vardy]

parameters: $d = 8$ $m = 4$ $\mathbf{s} = (0, 2, 4, 6)$, base field $\mathbb{F}_{97}$

input: $(24, 31, 15, 32, 83, 27, 20, 59)$ and $\mathbf{F} = [1 \;\; R \;\; R^2 \;\; R^3]^T$

iteration: $i = 4$ **point:** 24, 31, 15, 32, 83, 27, 20, 59

shift $[3 \;\; 3 \;\; 4 \;\; 6]$

basis
$$\begin{bmatrix} x^3 + 31x^2 + 27x + 3 & 36 & 0 & 0 \\ 54x^3 + 56x^2 + 56x + 36 & x + 65 & 0 & 0 \\ 56x^2 + 43x + 35 & 60 & 1 & 0 \\ 52x^2 + 33x + 60 & 68 & 0 & 1 \end{bmatrix}$$

values
$$\begin{bmatrix} 0 & 0 & 0 & 0 & 95 & 50 & 66 & 0 \\ 0 & 0 & 0 & 0 & 54 & 0 & 19 & 58 \\ 0 & 0 & 0 & 0 & 4 & 45 & 79 & 95 \\ 0 & 0 & 0 & 0 & 7 & 31 & 41 & 17 \end{bmatrix}$$

fast divide and conquer interpolation

iterative algorithm [van Barel-Bultheel / Beckermann-Labahn / Kötter-Vardy]

parameters: $d = 8$ $m = 4$ $\mathbf{s} = (0, 2, 4, 6)$, base field $\mathbb{F}_{97}$

input: $(24, 31, 15, 32, 83, 27, 20, 59)$ and $\mathbf{F} = [1 \ \ R \ \ R^2 \ \ R^3]^T$

iteration: $i = 5$ **point:** 24, 31, 15, 32, 83, 27, 20, 59

shift $[4 \quad 3 \quad 4 \quad 6]$

basis
$$\begin{bmatrix} x^4 + 45x^3 + 73x^2 + 90x + 42 & 36x + 19 & 0 & 0 \\ 81x^3 + 20x^2 + 9x + 20 & x + 67 & 0 & 0 \\ 2x^3 + 21x^2 + 41 & 35 & 1 & 0 \\ 52x^3 + 15x^2 + 79x + 22 & 0 & 0 & 1 \end{bmatrix}$$

values
$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 13 & 13 & 0 \\ 0 & 0 & 0 & 0 & 0 & 89 & 55 & 58 \\ 0 & 0 & 0 & 0 & 0 & 48 & 17 & 95 \\ 0 & 0 & 0 & 0 & 0 & 12 & 78 & 17 \end{bmatrix}$$

fast divide and conquer interpolation

iterative algorithm [van Barel-Bultheel / Beckermann-Labahn / Kötter-Vardy]

parameters: $d = 8$ $m = 4$ $\mathbf{s} = (0, 2, 4, 6)$, base field $\mathbb{F}_{97}$

input: $(24, 31, 15, 32, 83, 27, 20, 59)$ and $\mathbf{F} = [1 \;\; R \;\; R^2 \;\; R^3]^T$

iteration: $i = 6$ **point:** 24, 31, 15, 32, 83, 27, 20, 59

shift $[4 \quad 4 \quad 4 \quad 6]$

basis
$$\begin{bmatrix} x^4 + 19x^3 + 57x^2 + 44x + 26 & 74x + 43 & 0 & 0 \\ 81x^4 + 64x^3 + 51x^2 + 68x + 42 & x^2 + 40x + 34 & 0 & 0 \\ 3x^3 + 44x^2 + 54x + 64 & 6x + 49 & 1 & 0 \\ 28x^3 + 45x^2 + 44x + 52 & 50x + 52 & 0 & 1 \end{bmatrix}$$

values
$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 66 & 70 \\ 0 & 0 & 0 & 0 & 0 & 0 & 3 & 13 \\ 0 & 0 & 0 & 0 & 0 & 0 & 56 & 55 \\ 0 & 0 & 0 & 0 & 0 & 0 & 15 & 7 \end{bmatrix}$$

fast divide and conquer interpolation

iterative algorithm [van Barel-Bultheel / Beckermann-Labahn / Kötter-Vardy]

parameters: $d = 8$ $m = 4$ $\mathbf{s} = (0, 2, 4, 6)$, base field $\mathbb{F}_{97}$

input: $(24, 31, 15, 32, 83, 27, 20, 59)$ and $\mathbf{F} = [1 \;\; R \;\; R^2 \;\; R^3]^T$

iteration: $i = 7$ **point:** 24, 31, 15, 32, 83, 27, 20, 59

shift $[5 \quad 4 \quad 4 \quad 6]$

basis

$$\begin{bmatrix} x^5 + 96x^4 + 65x^3 + 68x^2 + 19x + 62 & 74x^2 + 18x + 13 & 0 & 0 \\ 6x^4 + 94x^3 + 44x^2 + 66x + 32 & x^2 + 19x + 10 & 0 & 0 \\ 55x^4 + 78x^3 + 75x^2 + 49x + 39 & 2x + 86 & 1 & 0 \\ 13x^4 + 81x^3 + 10x^2 + 34x + 2 & 42x + 29 & 0 & 1 \end{bmatrix}$$

values

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 14 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 25 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 44 \end{bmatrix}$$

fast divide and conquer interpolation

iterative algorithm [van Barel-Bultheel / Beckermann-Labahn / Kötter-Vardy]

parameters: $d = 8$ $m = 4$ $\mathbf{s} = (0, 2, 4, 6)$, base field $\mathbb{F}_{97}$

input: $(24, 31, 15, 32, 83, 27, 20, 59)$ and $\mathbf{F} = [1 \; R \; R^2 \; R^3]^T$

iteration: $i = 8$ **point:** 24, 31, 15, 32, 83, 27, 20, 59

shift $[5 \quad 5 \quad 4 \quad 6]$

basis
$$\begin{bmatrix} x^5 + 12x^4 + 10x^3 + 34x^2 + 65x + 2 & 60x^2 + 43x + 67 & 0 & 0 \\ 6x^5 + 31x^4 + 27x^3 + 89x^2 + 18x + 52 & x^3 + 57x^2 + 53x + 89 & 0 & 0 \\ 2x^4 + 56x^3 + 42x^2 + 48x + 15 & 72x^2 + 12x + 30 & 1 & 0 \\ 40x^4 + 19x^3 + 14x^2 + 40x + 49 & 53x^2 + 79x + 74 & 0 & 1 \end{bmatrix}$$

values
$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

fast divide and conquer interpolation

iterative algorithm [van Barel-Bultheel / Beckermann-Labahn / Kötter-Vardy]

parameters: $d = 8$ $\quad m = 4$ $\quad \mathbf{s} = (0, 2, 4, 6)$, base field $\mathbb{F}_{97}$

input: $(24, 31, 15, 32, 83, 27, 20, 59)$ and $\mathbf{F} = [1 \;\; R \;\; R^2 \;\; R^3]^T$

iteration: $i = 8$ **point:** 24, 31, 15, 32, 83, 27, 20, 59

shift $[5 \quad 5 \quad 4 \quad 6]$

basis

$$\begin{bmatrix} x^5 + 12x^4 + 10x^3 + 34x^2 + 65x + 2 & 60x^2 + 43x + 67 & 0 & 0 \\ 6x^5 + 31x^4 + 27x^3 + 89x^2 + 18x + 52 & x^3 + 57x^2 + 53x + 89 & 0 & 0 \\ 2x^4 + 56x^3 + 42x^2 + 48x + 15 & 72x^2 + 12x + 30 & 1 & 0 \\ 40x^4 + 19x^3 + 14x^2 + 40x + 49 & 53x^2 + 79x + 74 & 0 & 1 \end{bmatrix}$$

$$Q(x, y) = (2x^4 + 56x^3 + 42x^2 + 48x + 15) + (72x^2 + 12x + 30)y + y^2$$

values

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

fast divide and conquer interpolation

iterative algorithm [van Barel-Bultheel / Beckermann-Labahn / Kötter-Vardy]

input: vector $\mathbf{F} = \begin{bmatrix} f_1 \\ \vdots \\ f_m \end{bmatrix}$, points $\alpha_1, \ldots, \alpha_d \in \mathbb{K}$, shift $\mathbf{s} = (s_1, \ldots, s_m) \in \mathbb{Z}^m$

1. $\mathbf{P} = \begin{bmatrix} -\mathbf{p}_1- \\ \vdots \\ -\mathbf{p}_m- \end{bmatrix}$ = identity matrix in $\mathbb{K}[x]^{m \times m}$

2. for i from 1 to d:

- **a.** choose pivot π with smallest s_π such that $f_\pi(\alpha_i) \neq 0$
  update pivot shift $s_\pi = s_\pi + 1$
- **b.** constant elimination: for $j \neq \pi$ do $\mathbf{p}_j \leftarrow \mathbf{p}_j - \frac{f_j(\alpha_i)}{f_\pi(\alpha_i)} \mathbf{p}_\pi$
  polynomial elimination: $\mathbf{p}_\pi \leftarrow (x - \alpha_i)\mathbf{p}_\pi$
- **c.** compute residual equation: for $j \neq \pi$ do $f_j \leftarrow f_j - \frac{f_j(\alpha_i)}{f_\pi(\alpha_i)} f_\pi$
  $f_\pi \leftarrow (x - \alpha_i) f_\pi$

after i iterations: $\mathbf{P}$ is an $\mathbf{s}$-reduced basis of solutions for $(\alpha_1, \ldots, \alpha_i)$

fast divide and conquer interpolation

iterative algorithm: complexity aspects

at step i, $\mathbf{P}$ and $\mathbf{F}$ are left multiplied by $\mathbf{E}_i = \begin{bmatrix} \mathbf{I}_{\pi-1} & \boldsymbol{\lambda}_1 & \mathbf{0} \\ \mathbf{0} & x-\alpha & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\lambda}_2 & \mathbf{I}_{m-\pi} \end{bmatrix}$
where $\boldsymbol{\lambda}_1 \in \mathbb{K}^{(\pi-1)\times 1}$ and $\boldsymbol{\lambda}_2 \in \mathbb{K}^{(m-\pi)\times 1}$ are constant

fast divide and conquer interpolation

iterative algorithm: complexity aspects

at step i, $\mathbf{P}$ and $\mathbf{F}$ are left multiplied by $\mathbf{E}_i = \begin{bmatrix} \mathbf{I}_{\pi-1} & \boldsymbol{\lambda}_1 & \mathbf{0} \\ \mathbf{0} & x-\alpha & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\lambda}_2 & \mathbf{I}_{m-\pi} \end{bmatrix}$
where $\boldsymbol{\lambda}_1 \in \mathbb{K}^{(\pi-1)\times 1}$ and $\boldsymbol{\lambda}_2 \in \mathbb{K}^{(m-\pi)\times 1}$ are constant

complexity $O(m^2 d^2)$:

- iteration with d steps
- each step: evaluation of $\mathbf{F}$ + multiplications $\mathbf{E}_i\mathbf{F}$ and $\mathbf{E}_i\mathbf{P}$
- at any stage $\mathbf{P}$ has degree $\leqslant d$ and dimensions $m \times m$
- at any stage $\mathbf{F}$ has degree $< 2d$ and dimensions $m \times 1$

one gets $O(md^2)$ with either:
. normalizing at each step + finer analysis
. "balanced" input shift + finer analysis
(shifts in RS list-decoding are balanced)

fast divide and conquer interpolation

iterative algorithm: complexity aspects

at step i, $\mathbf{P}$ and $\mathbf{F}$ are left multiplied by $\mathbf{E}_i = \begin{bmatrix} \mathbf{I}_{\pi-1} & \boldsymbol{\lambda_1} & \mathbf{0} \\ \mathbf{0} & x-\alpha & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\lambda_2} & \mathbf{I}_{m-\pi} \end{bmatrix}$
where $\boldsymbol{\lambda_1} \in \mathbb{K}^{(\pi-1)\times 1}$ and $\boldsymbol{\lambda_2} \in \mathbb{K}^{(m-\pi)\times 1}$ are constant

complexity $O(m^2 d^2)$:

- iteration with d steps
- each step: evaluation of $\mathbf{F}$ + multiplications $\mathbf{E}_i\mathbf{F}$ and $\mathbf{E}_i\mathbf{P}$
- at any stage $\mathbf{P}$ has degree $\leqslant d$ and dimensions $m \times m$
- at any stage $\mathbf{F}$ has degree $< 2d$ and dimensions $m \times 1$

one gets $O(md^2)$ with either:
. normalizing at each step + finer analysis
. “balanced” input shift + finer analysis
(shifts in RS list-decoding are balanced)

correctness:

- the main task is to prove the base case ($d = 1$, single point)
- then, correctness follows from the “basis multiplication theorem”

fast divide and conquer interpolation

general multiplication-based approach for relations

algorithms based on polynomial matrix multiplication

[Beckermann-Labahn '94+'97] [Giorgi-Jeannerod-Villard 2003]

- compute a first basis $\mathbf{P}_1$ for a subproblem
- update the input instance to get the second subproblem
- compute a second basis $\mathbf{P}_2$ for this second subproblem
- the output basis of solutions is $\mathbf{P}_2\mathbf{P}_1$

we want $\mathbf{P}_2\mathbf{P}_1$ shifted reduced

$\mathbf{P}_2\mathbf{P}_1$ reduced not implied by "$\mathbf{P}_1$ reduced and $\mathbf{P}_2$ reduced"

fast divide and conquer interpolation

general multiplication-based approach for relations

algorithms based on polynomial matrix multiplication
[Beckermann-Labahn '94+'97] [Giorgi-Jeannerod-Villard 2003]

- compute a first basis $\mathbf{P}_1$ for a subproblem
- update the input instance to get the second subproblem
- compute a second basis $\mathbf{P}_2$ for this second subproblem
- the output basis of solutions is $\mathbf{P}_2\mathbf{P}_1$

we want $\mathbf{P}_2\mathbf{P}_1$ shifted reduced

$\mathbf{P}_2\mathbf{P}_1$ reduced not implied by "$\mathbf{P}_1$ reduced and $\mathbf{P}_2$ reduced"

theorem:
($\mathbf{P}_1$ is $\mathbf{s}$-reduced and $\mathbf{P}_2$ is $\mathbf{t}$-reduced") $\Rightarrow$ $\mathbf{P}_2\mathbf{P}_1$ is $\mathbf{s}$-reduced

where $\mathbf{t}$ is a shift trivially computed from $\mathbf{s}$ and $\mathbf{P}_1$ $\quad (\mathbf{t} = \mathrm{rdeg}_{\mathbf{s}}(\mathbf{P}_1))$

fast divide and conquer interpolation

bonus: detailed statement and proof

let $\mathcal{M} \subseteq \mathcal{M}_1$ be two $\mathbb{K}[x]$-submodules of $\mathbb{K}[x]^m$ of rank m,
let $\mathbf{P}_1 \in \mathbb{K}[x]^{m\times m}$ be a basis of $\mathcal{M}_1$,
let $\mathbf{s} \in \mathbb{Z}^m$ and $\mathbf{t} = \mathrm{rdeg}_{\mathbf{s}}(\mathbf{P}_1)$,

- the rank of the module $\mathcal{M}_2 = \{\boldsymbol{\lambda} \in \mathbb{K}[x]^{1\times m} \mid \boldsymbol{\lambda}\mathbf{P}_1 \in \mathcal{M}\}$ is m and for any basis $\mathbf{P}_2 \in \mathbb{K}[x]^{m\times m}$ of $\mathcal{M}_2$, the product $\mathbf{P}_2\mathbf{P}_1$ is a basis of $\mathcal{M}$
- if $\mathbf{P}_1$ is $\mathbf{s}$-reduced and $\mathbf{P}_2$ is $\mathbf{t}$-reduced, then $\mathbf{P}_2\mathbf{P}_1$ is $\mathbf{s}$-reduced

fast divide and conquer interpolation

bonus: detailed statement and proof

let $\mathcal{M} \subseteq \mathcal{M}_1$ be two $\mathbb{K}[x]$-submodules of $\mathbb{K}[x]^m$ of rank m,
let $\mathbf{P}_1 \in \mathbb{K}[x]^{m \times m}$ be a basis of $\mathcal{M}_1$,
let $s \in \mathbb{Z}^m$ and $t = \mathrm{rdeg}_s(\mathbf{P}_1)$,

- the rank of the module $\mathcal{M}_2 = \{\boldsymbol{\lambda} \in \mathbb{K}[x]^{1 \times m} \mid \boldsymbol{\lambda}\mathbf{P}_1 \in \mathcal{M}\}$ is m and for any basis $\mathbf{P}_2 \in \mathbb{K}[x]^{m \times m}$ of $\mathcal{M}_2$, the product $\mathbf{P}_2\mathbf{P}_1$ is a basis of $\mathcal{M}$
- if $\mathbf{P}_1$ is s-reduced and $\mathbf{P}_2$ is t-reduced, then $\mathbf{P}_2\mathbf{P}_1$ is s-reduced

Let $\mathbf{A} \in \mathbb{K}[x]^{m \times m}$ denote the adjugate of $\mathbf{P}_1$. Then, we have $\mathbf{A}\mathbf{P}_1 = \det(\mathbf{P}_1)\mathbf{I}_m$. Thus, $\mathbf{p}\mathbf{A}\mathbf{P}_1 = \det(\mathbf{P}_1)\mathbf{p} \in \mathcal{M}$ for all $\mathbf{p} \in \mathcal{M}$, and therefore $\mathcal{M}\mathbf{A} \subseteq \mathcal{M}_2$. Now, the nonsingularity of $\mathbf{A}$ ensures that $\mathcal{M}\mathbf{A}$ has rank m; this implies that $\mathcal{M}_2$ has rank m as well (see e.g. [Dummit-Foote 2004, Sec. 12.1, Thm. 4]). The matrix $\mathbf{P}_2\mathbf{P}_1$ is nonsingular since $\det(\mathbf{P}_2\mathbf{P}_1) \neq 0$. Now let $\mathbf{p} \in \mathcal{M}$; we want to prove that $\mathbf{p}$ is a $\mathbb{K}[x]$-linear combination of the rows of $\mathbf{P}_2\mathbf{P}_1$. First, $\mathbf{p} \in \mathcal{M}_1$, so there exists $\boldsymbol{\lambda} \in \mathbb{K}[x]^{1 \times m}$ such that $\mathbf{p} = \boldsymbol{\lambda}\mathbf{P}_1$. But then $\boldsymbol{\lambda} \in \mathcal{M}_2$, and thus there exists $\boldsymbol{\mu} \in \mathbb{K}[x]^{1 \times m}$ such that $\boldsymbol{\lambda} = \boldsymbol{\mu}\mathbf{P}_2$. This yields the combination $\mathbf{p} = \boldsymbol{\mu}\mathbf{P}_2\mathbf{P}_1$.

bonus: detailed statement and proof

let $\mathcal{M} \subseteq \mathcal{M}_1$ be two $\mathbb{K}[x]$-submodules of $\mathbb{K}[x]^m$ of rank m,
let $\mathbf{P}_1 \in \mathbb{K}[x]^{m \times m}$ be a basis of $\mathcal{M}_1$,
let $\mathbf{s} \in \mathbb{Z}^m$ and $\mathbf{t} = \mathrm{rdeg}_{\mathbf{s}}(\mathbf{P}_1)$,

- the rank of the module $\mathcal{M}_2 = \{\boldsymbol{\lambda} \in \mathbb{K}[x]^{1 \times m} \mid \boldsymbol{\lambda}\mathbf{P}_1 \in \mathcal{M}\}$ is m and for any basis $\mathbf{P}_2 \in \mathbb{K}[x]^{m \times m}$ of $\mathcal{M}_2$, the product $\mathbf{P}_2\mathbf{P}_1$ is a basis of $\mathcal{M}$
- if $\mathbf{P}_1$ is $\mathbf{s}$-reduced and $\mathbf{P}_2$ is $\mathbf{t}$-reduced, then $\mathbf{P}_2\mathbf{P}_1$ is $\mathbf{s}$-reduced

Let $\mathbf{d} = \mathrm{rdeg}_{\mathbf{t}}(\mathbf{P}_2)$; we have $\mathbf{d} = \mathrm{rdeg}_{\mathbf{s}}(\mathbf{P}_2\mathbf{P}_1)$ by the predictable degree property. Using $\mathbf{X}^{-\mathbf{d}}\mathbf{P}_2\mathbf{P}_1\mathbf{X}^{\mathbf{s}} = \mathbf{X}^{-\mathbf{d}}\mathbf{P}_2\mathbf{X}^{\mathbf{t}}\mathbf{X}^{-\mathbf{t}}\mathbf{P}_1\mathbf{X}^{\mathbf{s}}$, we obtain that $\mathrm{lm}_{\mathbf{s}}(\mathbf{P}_2\mathbf{P}_1) = \mathrm{lm}_{\mathbf{t}}(\mathbf{P}_2)\mathrm{lm}_{\mathbf{s}}(\mathbf{P}_1)$. By assumption, $\mathrm{lm}_{\mathbf{t}}(\mathbf{P}_2)$ and $\mathrm{lm}_{\mathbf{s}}(\mathbf{P}_1)$ are invertible, and therefore $\mathrm{lm}_{\mathbf{s}}(\mathbf{P}_2\mathbf{P}_1)$ is invertible as well; thus $\mathbf{P}_2\mathbf{P}_1$ is $\mathbf{s}$-reduced.

fast divide and conquer interpolation

divide and conquer algorithm [Beckermann-Labahn '94+'97]

input: $\mathbf{F}, (\alpha_1, \ldots, \alpha_d), \mathbf{s}$
output: $\mathbf{P}$

- if $d \leqslant \texttt{threshold}$: call iterative algorithm
- else:

  a. $G_1 \leftarrow (x - \alpha_1) \cdots (x - \alpha_{\lfloor d/2 \rfloor})$; $G_2 \leftarrow (x - \alpha_{\lfloor d/2 \rfloor + 1}) \cdots (x - \alpha_d)$

  b. $\mathbf{P}_1 \leftarrow$ recursive call on $\mathbf{F}$ rem $G_1, (\alpha_1, \ldots, \alpha_{\lfloor d/2 \rfloor}), \mathbf{s}$

  c. updated shift: $\mathbf{t} \leftarrow \mathrm{rdeg}_{\mathbf{s}}(\mathbf{P}_1)$

  d. residual equation: $\mathbf{F} \leftarrow \frac{1}{G_1}\mathbf{P}_1\mathbf{F}$

  e. $\mathbf{P}_2 \leftarrow$ recursive call $\mathbf{F}$ rem $G_2, (\alpha_{\lfloor d/2 \rfloor + 1}, \ldots, \alpha_d), \mathbf{t}$

  f. return the product $\mathbf{P}_2\mathbf{P}_1$

fast divide and conquer interpolation

divide and conquer algorithm [Beckermann-Labahn '94+'97]

input: $\mathbf{F}, (\alpha_1, \ldots, \alpha_d), \mathbf{s}$
output: $\mathbf{P}$

- if $d \leqslant \texttt{threshold}$: call iterative algorithm
- else:

  a. $G_1 \leftarrow (x - \alpha_1) \cdots (x - \alpha_{\lfloor d/2 \rfloor})$; $G_2 \leftarrow (x - \alpha_{\lfloor d/2 \rfloor + 1}) \cdots (x - \alpha_d)$

  b. $\mathbf{P}_1 \leftarrow$ recursive call on $\mathbf{F}$ rem $G_1, (\alpha_1, \ldots, \alpha_{\lfloor d/2 \rfloor}), \mathbf{s}$

  c. updated shift: $\mathbf{t} \leftarrow \mathrm{rdeg}_{\mathbf{s}}(\mathbf{P}_1)$

  d. residual equation: $\mathbf{F} \leftarrow \frac{1}{G_1} \mathbf{P}_1 \mathbf{F}$

  e. $\mathbf{P}_2 \leftarrow$ recursive call $\mathbf{F}$ rem $G_2, (\alpha_{\lfloor d/2 \rfloor + 1}, \ldots, \alpha_d), \mathbf{t}$

  f. return the product $\mathbf{P}_2 \mathbf{P}_1$

correctness:

- correctness of base case
- then, direct consequence of the "basis multiplication theorem"
- residual: $\{\mathbf{p} \mid \mathbf{p}\mathbf{P}_1\mathbf{F} = 0 \bmod G\} = \{\mathbf{p} \mid \mathbf{p}(\frac{1}{G_1}\mathbf{P}_1\mathbf{F}) = 0 \bmod G_2\}$

fast divide and conquer interpolation

divide and conquer algorithm [Beckermann-Labahn '94+'97]

input: $\mathbf{F}, (\alpha_1, \ldots, \alpha_d), \mathbf{s}$
output: $\mathbf{P}$

- if $d \leqslant \texttt{threshold}$: call iterative algorithm
- else:

  a. $G_1 \leftarrow (x - \alpha_1) \cdots (x - \alpha_{\lfloor d/2 \rfloor})$; $G_2 \leftarrow (x - \alpha_{\lfloor d/2 \rfloor + 1}) \cdots (x - \alpha_d)$
  b. $\mathbf{P}_1 \leftarrow$ recursive call on $\mathbf{F}$ rem $G_1, (\alpha_1, \ldots, \alpha_{\lfloor d/2 \rfloor}), \mathbf{s}$
  c. updated shift: $\mathbf{t} \leftarrow \mathrm{rdeg}_{\mathbf{s}}(\mathbf{P}_1)$
  d. residual equation: $\mathbf{F} \leftarrow \frac{1}{G_1}\mathbf{P}_1\mathbf{F}$
  e. $\mathbf{P}_2 \leftarrow$ recursive call $\mathbf{F}$ rem $G_2, (\alpha_{\lfloor d/2 \rfloor + 1}, \ldots, \alpha_d), \mathbf{t}$
  f. return the product $\mathbf{P}_2\mathbf{P}_1$

complexity $O(m^\omega M(d) \log(d))$**:**

- if $\omega = 2$, quasi-linear in worst-case output size
- most expensive step in the recursion is the product $\mathbf{P}_2\mathbf{P}_1$
- equation $\mathcal{C}(m, d) = \mathcal{C}(m, \lfloor d/2 \rfloor) + \mathcal{C}(m, \lceil d/2 \rceil) + O(m^\omega M(d))$

fast divide and conquer interpolation

divide and conquer: complexity aspects

input: $\deg(\mathbf{F}) < d$ output: $\deg(\mathbf{P}) \leqslant d$

complexity of each step:

- residual $\mathbf{F} \leftarrow \frac{1}{M_1}\mathbf{P}_1\mathbf{F}$ $O(m^2 M(d))$
- $\mathbf{F}$ rem M_1 and $\mathbf{F}$ rem M_2 $O(m M(d))$
- product $\mathbf{P}_2\mathbf{P}_1$ $O(m^\omega M(d))$
- two recursive calls $2\mathcal{C}(m, \lfloor d/2 \rceil)$

input: $\deg(\mathbf{F}) < d$ output: $\deg(\mathbf{P}) \leqslant d$

complexity of each step:

- residual $\mathbf{F} \leftarrow \frac{1}{M_1}\mathbf{P}_1\mathbf{F}$ — $O(m^2 M(d))$
- $\mathbf{F}$ rem M_1 and $\mathbf{F}$ rem M_2 — $O(m M(d))$
- product $\mathbf{P}_2\mathbf{P}_1$ — $O(m^\omega M(d))$
- two recursive calls — $2\mathcal{C}(m, \lfloor d/2 \rceil)$

$$\begin{cases} \mathcal{C}(m, d) = \mathcal{C}(m, \lfloor d/2 \rfloor) + \mathcal{C}(m, \lceil d/2 \rceil) + O(m^\omega M(d)) \\ d \text{ base cases, each one costs } O(m) \end{cases}$$

$$\Rightarrow \quad O(m^\omega M(d) \log(d))$$

unrolling: $m^\omega \left(M(d) + 2M(\frac{d}{2}) + 4M(\frac{d}{4}) + \cdots + \frac{d}{2}M(2)\right) + dm$

divide and conquer: complexity aspects

input: $\deg(\mathbf{F}) < d$ output: $\deg(\mathbf{P}) \leqslant d$

complexity of each step:

- residual $\mathbf{F} \leftarrow \frac{1}{M_1}\mathbf{P}_1\mathbf{F}$ — $O(m^2 M(d))$
- $\mathbf{F}$ rem M_1 and $\mathbf{F}$ rem M_2 — $O(m M(d))$
- product $\mathbf{P}_2\mathbf{P}_1$ — $O(m^\omega M(d))$
- two recursive calls — $2\mathcal{C}(m, \lfloor d/2 \rceil)$

output: $\deg(\mathbf{P}) \approx \lceil \frac{d}{m} \rceil$

$s = 0$ and generic $\mathbf{F}$:

$O(m^\omega M(\lceil \frac{d}{m} \rceil))$

unchanged

$O(m^\omega M(\lceil \frac{d}{m} \rceil))$

unchanged

- partial linearization

$$\begin{cases} \mathcal{C}(m, d) = \mathcal{C}(m, \lfloor d/2 \rfloor) + \mathcal{C}(m, \lceil d/2 \rceil) + O(m^\omega M(d)) \\ d \text{ base cases, each one costs } O(m) \end{cases}$$

$$\Rightarrow \quad O(m^\omega M(d) \log(d))$$

fast divide and conquer interpolation

divide and conquer: complexity aspects

input: $\deg(\mathbf{F}) < d$ output: $\deg(\mathbf{P}) \leqslant d$

complexity of each step:

- residual $\mathbf{F} \leftarrow \frac{1}{M_1}\mathbf{P}_1\mathbf{F}$ — $O(m^2 M(d))$
- $\mathbf{F}$ rem M_1 and $\mathbf{F}$ rem M_2 — $O(mM(d))$
- product $\mathbf{P}_2\mathbf{P}_1$ — $O(m^\omega M(d))$
- two recursive calls — $2\mathcal{C}(m, \lfloor d/2 \rceil)$

output: $\deg(\mathbf{P}) \approx \lceil \frac{d}{m} \rceil$

$s = 0$ and generic $\mathbf{F}$:

$O(m^\omega M(\lceil \frac{d}{m} \rceil))$

unchanged

$O(m^\omega M(\lceil \frac{d}{m} \rceil))$

unchanged

- partial linearization
- base case for $d \approx m$, costs $O(m^\omega)$

$$\begin{cases} \mathcal{C}(m, d) = \mathcal{C}(m, \lfloor d/2 \rfloor) + \mathcal{C}(m, \lceil d/2 \rceil) + O(m^\omega M(d)) \\ d \text{ base cases, each one costs } O(m) \end{cases}$$

$$\Rightarrow \quad O(m^\omega M(d) \log(d)) \qquad O(m^\omega M(\lceil \tfrac{d}{m} \rceil) \log(\lceil \tfrac{d}{m} \rceil))$$

fast divide and conquer interpolation

divide and conquer: complexity aspects

input: $\deg(\mathbf{F}) < d$ output: $\deg(\mathbf{P}) \leqslant d$

complexity of each step:

- residual $\mathbf{F} \leftarrow \frac{1}{M_1}\mathbf{P}_1\mathbf{F}$ $O(m^2 M(d))$
- $\mathbf{F}$ rem M_1 and $\mathbf{F}$ rem M_2 $O(mM(d))$
- product $\mathbf{P}_2\mathbf{P}_1$ $O(m^\omega M(d))$
- two recursive calls $2\mathcal{C}(m, \lfloor d/2 \rceil)$

output: $\deg(\mathbf{P}) \approx \lceil \frac{d}{m} \rceil$

$s = 0$ and generic $\mathbf{F}$:

$O(m^\omega M(\lceil \frac{d}{m} \rceil))$

unchanged

$O(m^\omega M(\lceil \frac{d}{m} \rceil))$

unchanged

- partial linearization
- base case for $d \approx m$, costs $O(m^\omega)$

$$\begin{cases} \mathcal{C}(m, d) = \mathcal{C}(m, \lfloor d/2 \rfloor) + \mathcal{C}(m, \lceil d/2 \rceil) + O(m^\omega M(d)) \\ d \text{ base cases, each one costs } O(m) \end{cases}$$

$$\Rightarrow \quad O(m^\omega M(d) \log(d)) \qquad O(m^\omega M(\lceil \tfrac{d}{m} \rceil) \log(\lceil \tfrac{d}{m} \rceil))$$

m	n	d	PM-Basis	PM-Basis with linearization
4	1	65536	1.6693	**1.26891**
16	1	16384	1.8535	**0.89652**
64	1	2048	2.2865	**0.14362**
256	1	1024	36.620	**0.20660**

overview of the state of the art:

- recursive algorithm: from [Beckermann-Labahn 1994] (for Hermite-Padé)
  it also works for $\mathbf{F} \in \mathbb{K}[x]^{m \times n}$ with $n > 1$

- [Giorgi-Jeannerod-Villard 2003] achieved $O(m^\omega M(d) \log(d))$
for $\mathbf{F}$ mod x^d, with $n \geqslant 1$ and $n \in O(m)$

- for $s = \mathbf{0}$ and generic $\mathbf{F}$: $\tilde{O}(m^\omega \lceil \frac{nd}{m} \rceil)$ [Lecerf, ca 2001, unpublished]

overview of the state of the art:

- recursive algorithm: from [Beckermann-Labahn 1994] (for Hermite-Padé)
  it also works for $\mathbf{F} \in \mathbb{K}[x]^{m \times n}$ with $n > 1$

- [Giorgi-Jeannerod-Villard 2003] achieved $O(m^\omega \mathsf{M}(d) \log(d))$
for $\mathbf{F}$ mod x^d, with $n \geqslant 1$ and $n \in O(m)$

- for $\mathbf{s} = \mathbf{0}$ and generic $\mathbf{F}$: $O\tilde{~}(m^\omega \lceil \frac{nd}{m} \rceil)$ [Lecerf, ca 2001, unpublished]

- more recently: $O\tilde{~}(m^{\omega-1} nd)$ for $\mathbf{F}$ mod x^d
[Storjohann 2006] [Zhou-Labahn 2012] [Jeannerod-Neiger-Villard 2020]
$\rightsquigarrow$ any $\mathbf{s}$, no genericity assumption, returns the canonical $\mathbf{s}$-Popov basis

fast divide and conquer interpolation

vector rational interpolation: recent progress

overview of the state of the art:

- recursive algorithm: from [Beckermann-Labahn 1994] (for Hermite-Padé)
  it also works for $\mathbf{F} \in \mathbb{K}[x]^{m \times n}$ with $n > 1$

- [Giorgi-Jeannerod-Villard 2003] achieved $O(m^{\omega} M(d) \log(d))$
for $\mathbf{F} \bmod x^d$, with $n \geqslant 1$ and $n \in O(m)$

- for $s = \mathbf{0}$ and generic $\mathbf{F}$: $O\tilde{~}(m^{\omega} \lceil \frac{nd}{m} \rceil)$ [Lecerf, ca 2001, unpublished]

- more recently: $O\tilde{~}(m^{\omega-1} nd)$ for $\mathbf{F} \bmod x^d$
[Storjohann 2006] [Zhou-Labahn 2012] [Jeannerod-Neiger-Villard 2020]
$\rightsquigarrow$ any s, no genericity assumption, returns the canonical s-Popov basis

- $\mathbf{F}$ mod G and general modular matrix equations in similar complexity
[Beckermann-Labahn 1997] [Jeannerod-Neiger-Schost-Villard 2017]
[Neiger-Vu 2017] [Rosenkilde-Storjohann 2021]
$\rightsquigarrow$ any s, no genericity assumption, returns the canonical s-Popov basis

outline

- **computer algebra**
  - efficient algorithms and software
  - for matrices over a field
  - for univariate polynomials
- **Reed-Solomon decoding**
  - context and unique decoding problem
  - key equations and how to solve them
  - correcting more errors?
- **polynomial matrices**
  - introduction to vector interpolation
  - core algorithms & shifted normal forms
  - fast divide and conquer interpolation
- **efficient list decoding**

outline

computer algebra

- efficient algorithms and software
- for matrices over a field
- for univariate polynomials

Reed-Solomon decoding

- context and unique decoding problem
- key equations and how to solve them
- correcting more errors?

polynomial matrices

- introduction to vector interpolation
- core algorithms & shifted normal forms
- fast divide and conquer interpolation

efficient list decoding

- the Guruswami-Sudan algorithm
- via structured systems or basis reduction
- a word on extensions

list decoding problem

for convenience, we use the agreement parameter $t = n - e$:

$$\#\{i \mid w(\alpha_i) \neq \beta_i\} \leqslant e \quad \Leftrightarrow \quad \#\{i \mid w(\alpha_i) = \beta_i\} \geqslant t$$

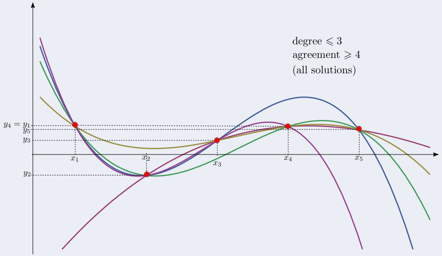
input:

- $\alpha_1, \ldots, \alpha_n$ the n distinct evaluation points in $\mathbb{K}$,
- k the degree bound, $t = n - e$ the agreement,
- $(\beta_1, \ldots, \beta_n)$ the received word in $\mathbb{K}^n$

list decoding requirement: $t^2 > kn$ [Guruswami-Sudan'99]

output: **all** polynomials $w(x)$ in $\mathbb{K}[x]$ such that

$$\deg(w) \leqslant k \qquad \text{and} \qquad \#\{i \mid w(\alpha_i) = \beta_i\} \geqslant t$$



list decoding problem

for convenience, we use the agreement parameter $t = n - e$:
$\#\{i \mid w(\alpha_i) \neq \beta_i\} \leqslant e \quad \Leftrightarrow \quad \#\{i \mid w(\alpha_i) = \beta_i\} \geqslant t$

input:

- $\alpha_1, \ldots, \alpha_n$ the n distinct evaluation points in $\mathbb{K}$,
- k the degree bound, $t = n - e$ the agreement,
- $(\beta_1, \ldots, \beta_n)$ the received word in $\mathbb{K}^n$

list decoding requirement: $t^2 > kn$ [Guruswami-Sudan'99]

output: **all** polynomials $w(x)$ in $\mathbb{K}[x]$ such that
$\deg(w) \leqslant k$ and $\#\{i \mid w(\alpha_i) = \beta_i\} \geqslant t$

Guruswami-Sudan algorithm:

- **interpolation step**
compute $Q(x, y)$ such that: $w(x)$ solution $\Rightarrow Q(x, w(x)) = 0$
- **root-finding step**
compute all y-roots of $Q(x, y)$, keep those that are solutions

introducing the interpolation+root-finding approach

consider **one** solution w_1:

key equation:

$$\Lambda_1 R \;=\; \Lambda_1 w_1 \quad \bmod G$$

where $R(\alpha_i) = \beta_i$, $\quad G(x) = \prod_{1 \leqslant i \leqslant n}(x - \alpha_i) \quad \Lambda_1(x) = \prod_{i \mid \text{error}_1}(x - \alpha_i)$

↓

obstacle: no uniqueness of solution $\frac{\mu_1}{\Lambda_1}$ for rational reconstruction

$$\Lambda_1 R \;=\; \mu_1 \quad \bmod G$$

with $\deg \mu_1 \leqslant e + k$

since $e \geqslant \frac{n-k}{2} \Rightarrow$ (unique decoding bound not satisfied),
possibly $\deg(\Lambda_1) + \deg(\Lambda_1 w_1) \;\geqslant\; n = \deg G$
(more unknowns than equations in the linearized problem)

introducing the interpolation+root-finding approach

note $\Lambda_1(R - w_1) = 0 \bmod G$, and consider **a second** solution w_2:

"extended" key equation:

$$\Lambda(R - w_1)(R - w_2) \;=\; 0 \quad \bmod G$$

where $\Lambda = \prod_{i \mid \text{error}_{1\wedge 2}} (x - \alpha_i) = \gcd(\Lambda_1, \Lambda_2)$

w_1 and w_2 are y-roots of the bivariate polynomial

$$Q(x,y) \;=\; \Lambda(y - w_1)(y - w_2) \;=\; \Lambda w_1 w_2 \;-\; \Lambda(w_1 + w_2)y \;+\; \Lambda y^2$$

$\rightsquigarrow$ similar remark for all ℓ solutions $w_1, \ldots, w_\ell$

properties of $Q(x,y)$:

- degree in y is ℓ = number of solutions
- weighted-degree $\deg_x(Q(x, x^k y))$ close to ℓk
- $Q(\alpha_i, \beta_i) = 0$ for every i (i.e. $Q(x, R) = 0 \bmod G$)

the Guruswami-Sudan algorithm

bivariate interpolation with multiplicities:

Input:

n points $\{(\alpha_i, \beta_i)\}_{1 \leqslant i \leqslant n}$ in $\mathbb{K}^2$, with the α_i's distinct

k the degree constraint, t the agreement

ℓ the list-size, s the multiplicity $(s \leqslant \ell)$

Output:

a nonzero polynomial $Q(x, y)$ in $\mathbb{K}[x, y]$ such that

(i)	$\deg_y(Q) \leqslant \ell$	(list-size condition)
(ii)	$\deg_x(Q(x, x^k y) < st$	(weighted-degree condition)
(iii)	$\forall i,\ Q(\alpha_i, \beta_i) = 0$ with multiplicity s	(vanishing condition)

the Guruswami-Sudan algorithm

bivariate interpolation with multiplicities:

Input:

n points $\{(\alpha_i, \beta_i)\}_{1 \leqslant i \leqslant n}$ in $\mathbb{K}^2$, with the α_i's distinct
k the degree constraint, t the agreement
ℓ the list-size, s the multiplicity $(s \leqslant \ell)$

Output:

a nonzero polynomial $Q(x, y)$ in $\mathbb{K}[x, y]$ such that

(i)	$\deg_y(Q) \leqslant \ell$	(list-size condition)
(ii)	$\deg_x(Q(x, x^k y) < st$	(weighted-degree condition)
(iii)	$\forall i,\ Q(\alpha_i, \beta_i) = 0$ with multiplicity s	(vanishing condition)

- find parameters ℓ and s
- **interpolation step**
compute $Q(x, y)$ such that: $w(x)$ solution $\Rightarrow Q(x, w(x)) = 0$
- **root-finding step**
compute all y-roots of $Q(x, y)$, keep those that are solutions

the Guruswami-Sudan algorithm

(i)	$\deg_y(Q) \leqslant \ell$	(list-size condition)
(ii)	$\deg_x(Q(x, x^k y) < st$	(weighted-degree condition)
(iii)	$\forall i,\ Q(\alpha_i, \beta_i) = 0$ with multiplicity s	(vanishing condition)

- find parameters ℓ and s
- **interpolation step**
compute $Q(x, y)$ such that: $w(x)$ solution $\Rightarrow Q(x, w(x)) = 0$
- **root-finding step**
compute all y-roots of $Q(x, y)$, keep those that are solutions

the Guruswami-Sudan algorithm

(i)	$\deg_y(Q) \leqslant \ell$	(list-size condition)
(ii)	$\deg_x(Q(x, x^k y) < st$	(weighted-degree condition)
(iii)	$\forall i,\ Q(\alpha_i, \beta_i) = 0$ with multiplicity s	(vanishing condition)

$w(x)$ solution

$\deg(w) \leqslant k$ $\xrightarrow{\text{(ii)}}$ $\deg Q(x, w(x)) < st$

$\#\{i \mid w(\alpha_i) = \beta_i\} \geqslant t$ $\xrightarrow{\text{(iii)}}$ $Q(x, w(x))$ has $\geqslant st$ roots

$Q(x, w(x)) = 0$

- find parameters ℓ and s
- **interpolation step**
  compute $Q(x, y)$ such that: $w(x)$ solution $\Rightarrow Q(x, w(x)) = 0$
- **root-finding step**
  compute all y-roots of $Q(x, y)$, keep those that are solutions

the Guruswami-Sudan algorithm

(i)	$\deg_y(Q) \leqslant \ell$	(list-size condition)
(ii)	$\deg_x(Q(x, x^k y) < st$	(weighted-degree condition)
(iii)	$\forall i,\ Q(\alpha_i, \beta_i) = 0$ with multiplicity s	(vanishing condition)

- list-size condition allows to work with polynomial matrices

identification $\mathbb{K}[x, y]_{\deg_y \leqslant \ell} \longleftrightarrow \mathbb{K}[x]^\ell$

$Q(x, y) = Q_0(x) + Q_1(x)y + \cdots + Q_\ell(x)y^\ell$

- weighted-degree condition handled via shifted forms

degree constraints $\deg(Q_j(x)) < st - jk$ for $j = 0, \ldots, \ell$

- find parameters ℓ and s
- **interpolation step**

compute $Q(x, y)$ such that: $w(x)$ solution $\Rightarrow Q(x, w(x)) = 0$

- **root-finding step**

compute all y-roots of $Q(x, y)$, keep those that are solutions

the Guruswami-Sudan algorithm

(i)	$\deg_y(Q) \leqslant \ell$	(list-size condition)
(ii)	$\deg_x(Q(x, x^k y) < st$	(weighted-degree condition)
(iii)	$\forall i,\ Q(\alpha_i, \beta_i) = 0$ with multiplicity s	(vanishing condition)

root-finding step: quasi-linear complexity

[Alekhnovich 2005] [Neiger-Rosenkilde-Schost 2017]

fastest known interpolation step: via univariate relations $O\tilde{~}(\ell^{\omega-1}s^2n)$

[Jeannerod-Neiger-Schost-Villard 2017]

- Sudan case ($s = 1$): vector rational interpolation
- general case: similar problem with s equations,

which have respective moduli G^s, G^{s-1}, ..., G

- find parameters ℓ and s
- **interpolation step**

compute $Q(x, y)$ such that: $w(x)$ solution $\Rightarrow Q(x, w(x)) = 0$

- **root-finding step**

compute all y-roots of $Q(x, y)$, keep those that are solutions

alternative approach: structured linear algebra

features common to all algorithms:

- use $(i) + (ii)$ to fix the linear unknowns:

$$Q = \sum_{0 \leqslant j \leqslant \ell} \sum_{0 \leqslant i < st - jk} q_{i,j} x^i y^j$$

- same number of linear unknowns: $(\ell + 1)st - \frac{\ell(\ell+1)}{2}k$
- same number of linear equations: $\frac{s(s+1)}{2}n$
- call a structured linear system solver

alternative approach: structured linear algebra

features common to all algorithms:

- use $(i) + (ii)$ to fix the linear unknowns:

$$Q = \sum_{0 \leqslant j \leqslant \ell} \sum_{0 \leqslant i < st - jk} q_{i,j} x^i y^j$$

- same number of linear unknowns: $(\ell + 1)st - \frac{\ell(\ell+1)}{2}k$
- same number of linear equations: $\frac{s(s+1)}{2}n$
- call a structured linear system solver

$$\begin{bmatrix} Q_0(x) & Q_1(x) \end{bmatrix} \begin{bmatrix} 2x^7 + 2x^6 + 5x^4 + 2x^2 + 4 \\ -1 \end{bmatrix} = 0 \bmod x^8$$

$$\left[\begin{array}{cccccc|ccc} q_{00} & q_{01} & q_{02} & q_{03} & q_{04} & q_{05} & q_{10} & q_{11} & q_{12} \end{array}\right] \left[\begin{array}{cccccccc} 4 & 0 & 2 & 0 & 5 & 0 & 2 & 2 \\ & 4 & 0 & 2 & 0 & 5 & 0 & 2 \\ & & 4 & 0 & 2 & 0 & 5 & 0 \\ & & & 4 & 0 & 2 & 0 & 5 \\ & & & & 4 & 0 & 2 & 0 \\ & & & & & 4 & 0 & 2 \\ \hline 6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ & 6 & 0 & 0 & 0 & 0 & 0 & 0 \\ & & 6 & 0 & 0 & 0 & 0 & 0 \end{array}\right] = 0$$

alternative approach: structured linear algebra

features common to all algorithms:

- use $(i) + (ii)$ to fix the linear unknowns:
  $$Q = \sum_{0 \leqslant j \leqslant \ell} \sum_{0 \leqslant i < st - jk} q_{i,j} x^i y^j$$
- same number of linear unknowns: $(\ell + 1)st - \frac{\ell(\ell+1)}{2} k$
- same number of linear equations: $\frac{s(s+1)}{2} n$
- call a structured linear system solver

$Q(x, y) = q_{00} + q_{01}x + q_{02}x^2 + q_{03}x^3 + q_{04}x^4 + (q_{10} + q_{11}x + q_{12}x^2)y + q_{20}y^2$:

$$\left[\begin{array}{ccccc:ccc:c} q_{00} & q_{01} & q_{02} & q_{03} & q_{04} & q_{10} & q_{11} & q_{12} & q_{20} \end{array}\right] \left[\begin{array}{cccc} 1 & 1 & \cdots & 1 \\ \alpha_1 & \alpha_2 & \cdots & \alpha_8 \\ \alpha_1^2 & \alpha_2^2 & \cdots & \alpha_8^2 \\ \alpha_1^3 & \alpha_2^3 & \cdots & \alpha_8^3 \\ \alpha_1^4 & \alpha_2^4 & \cdots & \alpha_8^4 \\ \hdashline \beta_1 & \beta_2 & \cdots & \beta_8 \\ \alpha_1\beta_1 & \alpha_2\beta_2 & \cdots & \alpha_8\beta_8 \\ \alpha_1^2\beta_1 & \alpha_2^2\beta_2 & \cdots & \alpha_8^2\beta_8 \\ \hdashline \beta_1^2 & \beta_2^2 & \cdots & \beta_8^2 \end{array}\right] = 0$$

alternative approach: structured linear algebra

Vandermonde-like system $O(\ell s^4 n^2)$

- [Olshevsky-Shokrollahi'99]
- linearize the vanishing condition on each point

alternative approach: structured linear algebra

Vandermonde-like system $O(\ell s^4 n^2)$

- [Olshevsky-Shokrollahi'99]
- linearize the vanishing condition on each point

Mosaic-Hankel system $O(\ell s^4 n^2)$

- [Roth-Ruckenstein'00] [Zeh-Gentner-Augot 2011]
- linearize the reversed extended key equation
- uses an adapted [Feng-Tzeng'91] solver

alternative approach: structured linear algebra

Vandermonde-like system $O(\ell s^4 n^2)$

- [Olshevsky-Shokrollahi'99]
- linearize the vanishing condition on each point

Mosaic-Hankel system $O(\ell s^4 n^2)$

- [Roth-Ruckenstein'00] [Zeh-Gentner-Augot 2011]
- linearize the reversed extended key equation
- uses an adapted [Feng-Tzeng'91] solver

Toeplitz-like system $O\tilde{}(\ell^{\omega-1} s^2 n)$

- [Chowdhury-Jeannerod-Neiger-Schost-Villard 2015]
- linearize the extended key equation
- uses the solver of [Bostan-Jeannerod-Schost 2007]

Las Vegas randomized

alternative approach: basis reduction

features common to all algorithms:

- use (i) to fix the polynomial unknowns:

  $$Q = \sum_{0 \leqslant j \leqslant \ell} Q_j(x) y^j \quad \longleftrightarrow \quad [Q_0(x) \cdots Q_\ell(x)]$$

- consider same interpolant $\mathbb{K}[x]$-module:

  $$\{Q \mid (i) + (iii)\} = \{\sum_{0 \leqslant j \leqslant \ell} Q_j(x) y^j \mid Q(\alpha_i, \beta_i) = 0 \text{ with mult. } s\}$$

- use (iii) to derive a basis of the module:

  $$\{Q \mid (i) + (iii)\} = \langle p_0(x,y), p_1(x,y), \ldots, p_\ell(x,y) \rangle$$

- call a $\mathbb{K}[x]$-module basis reduction algorithm,
  using a shift to satisfy the weighted-degree condition (ii)

alternative approach: basis reduction

features common to all algorithms:

- use (i) to fix the polynomial unknowns:

$$Q = \sum_{0 \leqslant j \leqslant \ell} Q_j(x) y^j \quad \longleftrightarrow \quad [Q_0(x) \cdots Q_\ell(x)]$$

- consider same interpolant $\mathbb{K}[x]$-module:

$$\{Q \mid (\mathrm{i}) + (\mathrm{iii})\} = \{\sum_{0 \leqslant j \leqslant \ell} Q_j(x) y^j \mid Q(\alpha_i, \beta_i) = 0 \text{ with mult. } s\}$$

- use (iii) to derive a basis of the module:

$$\{Q \mid (\mathrm{i}) + (\mathrm{iii})\} = \langle p_0(x,y), p_1(x,y), \ldots, p_\ell(x,y) \rangle$$

- call a $\mathbb{K}[x]$-module basis reduction algorithm,
  using a shift to satisfy the weighted-degree condition (ii)

$$\begin{array}{r} G \longrightarrow \\ y - R \longrightarrow \\ y(y-R) \longrightarrow \\ y^2(y-R) \longrightarrow \\ \vdots \\ y^{\ell-1}(y-R) \longrightarrow \end{array} \quad \begin{bmatrix} G & 0 & 0 & 0 & \cdots & 0 \\ -R & 1 & 0 & 0 & \cdots & 0 \\ 0 & -R & 1 & 0 & \cdots & 0 \\ 0 & 0 & -R & 1 & \cdots & 0 \\ \vdots & & \ddots & \ddots & \ddots & \\ 0 & \cdots & \cdots & 0 & -R & 1 \end{bmatrix}$$

alternative approach: basis reduction

features common to all algorithms:

- use (i) to fix the polynomial unknowns:

$$Q = \sum_{0\leqslant j\leqslant \ell} Q_j(x)y^j \quad \longleftrightarrow \quad [Q_0(x) \cdots Q_\ell(x)]$$

- consider same interpolant $\mathbb{K}[x]$-module:

$$\{Q \mid (i) + (iii)\} = \{\sum_{0\leqslant j\leqslant \ell} Q_j(x)y^j \mid Q(\alpha_i, \beta_i) = 0 \text{ with mult. } s\}$$

- use (iii) to derive a basis of the module:

$$\{Q \mid (i) + (iii)\} = \langle p_0(x,y), p_1(x,y), \ldots, p_\ell(x,y)\rangle$$

- call a $\mathbb{K}[x]$-module basis reduction algorithm,
  using a shift to satisfy the weighted-degree condition (ii)

$$\begin{array}{r} G \longrightarrow \\ y - R \longrightarrow \\ y^2 - R^2 \longrightarrow \\ y^3 - R^3 \longrightarrow \\ \vdots \\ y^\ell - R^\ell \longrightarrow \end{array} \quad \begin{bmatrix} G & 0 & 0 & 0 & \cdots & 0 \\ -R & 1 & 0 & 0 & \cdots & 0 \\ -R^2 & 0 & 1 & 0 & \cdots & 0 \\ -R^3 & 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & & \ddots & \ddots & \\ -R^\ell & 0 & \cdots & 0 & 0 & 1 \end{bmatrix}$$

alternative approach: basis reduction

basis reduction ≈ [Mulders-Storjohann 2003] quadratic in n

- [Reinhard 2003] $O(\ell^3 m^2 n^2)$
- [Lee-O'Sullivan 2008] $O(\ell^4 m n^2)$
- [Trifonov 2010] $O(m^3 n^2)$ (heuristic)

alternative approach: basis reduction

basis reduction ≈ [Mulders-Storjohann 2003] — quadratic in n

- [Reinhard 2003] — $O(\ell^3 m^2 n^2)$
- [Lee-O'Sullivan 2008] — $O(\ell^4 m n^2)$
- [Trifonov 2010] — $O(m^3 n^2)$ (heuristic)

basis reduction = matrix-half-GCD — ~linear in n

- [Alekhnovich 2002+2005] — $\tilde{O}(\ell^4 m^4 n)$

basis reduction = [Giorgi-Jeannerod-Villard 2003] — ~linear in n

- [Beelen-Brander 2010] — $\tilde{O}(\ell^4 m n)$
- [Bernstein 2010] — $\tilde{O}(\ell^{\omega+1} n)$
- [Cohn-Heninger 2011+2015] — $\tilde{O}(\ell^{\omega} m n)$

alternative approach: basis reduction

basis reduction ≈ [Mulders-Storjohann 2003] quadratic in n

- [Reinhard 2003] $O(\ell^3 m^2 n^2)$
- [Lee-O'Sullivan 2008] $O(\ell^4 m n^2)$
- [Trifonov 2010] $O(m^3 n^2)$ (heuristic)

basis reduction = matrix-half-GCD ˜linear in n

- [Alekhnovich 2002+2005] $O\tilde{}(\ell^4 m^4 n)$

basis reduction = [Giorgi-Jeannerod-Villard 2003] ˜linear in n

- [Beelen-Brander 2010] $O\tilde{}(\ell^4 m n)$
- [Bernstein 2010] $O\tilde{}(\ell^{\omega+1} n)$
- [Cohn-Heninger 2011+2015] $O\tilde{}(\ell^\omega m n)$

basis reduction = fastest known $O\tilde{}(\ell^{\omega-1} s^2 n)$

- [Neiger 2016] [Neiger-Vu 2017]
- do not go this way!

⇝ here, better call fast vector interpolation directly

generalizations of the interpolation step

summary for [Sudan '97] [Guruswami-Sudan '99]:

- list-decoding of Reed-Solomon codes, extends error-correction bound

compute $Q(x, y) = Q_0 + Q_1 y + \cdots + Q_m y^\ell$ such that

- $[Q_0, \ldots, Q_\ell]$ has small shifted degree
- $Q(\alpha_i, \beta_i) = 0$ with multiplicity μ for all i

generalizations of the interpolation step

[Kötter-Vardy 2003]
soft-decision decoding of Reed-Solomon codes

$\alpha_1, \ldots, \alpha_n$ are not pairwise distinct
compute $Q(x,y) = Q_0 + Q_1 y + \cdots + Q_\ell y^\ell$ such that

- $[Q_0, \ldots, Q_\ell]$ has small shifted degree
- $Q(\alpha_i, \beta_i) = 0$ with multiplicity μ_i for all i

generalizations of the interpolation step

[Guruswami-Rudra 2006]
list-decoding of folded Reed-Solomon codes:
further extends the error-correction bound up to the information-theoretic limit

[Devet-Goldberg-Heninger 2012]
Optimally robust Private Information Retrieval

compute $Q(x, y_1, \ldots, y_s) = \sum_{(j_1,\ldots,j_s)\in\Gamma} Q_{j_1,\ldots,j_s} y_1^{j_1} \cdots y_s^{j_s}$ such that

- $[Q_{j_1,\ldots,j_s}]_{(j_1,\ldots,j_s)\in\Gamma}$ has small shifted degree
- $Q(\alpha_i, \beta_{i1}, \ldots, \beta_{is}) = 0$ with multiplicity μ for all i

generalizations of the interpolation step

[Beelen-Rosenkilde-Solomatov 2022]
[Beelen-Neiger (preprint) 2023]
Guruswami-Sudan algorithm in the algebraic-geometry code setting

up to more precomputations, very similar context:

. . . also up to many technical details

$$\mathcal{M}_{s,\ell,\beta} = \Big\{ Q = \sum_{t=0}^{\ell} z^t Q_t \in F[z] \;\Big|\; Q_t \in \Delta(-tG),$$
$$Q \text{ has a root of multiplicity at least } s \text{ at } (P_j, \beta_j) \text{ for all } j \Big\}.$$

$$\mathcal{M}_{s,\ell,\beta} = \bigoplus_{t=0}^{s-1} (z - R)^t \Delta(G_t) \oplus \bigoplus_{t=s}^{\ell} f_t(z)(z-R)^s \Delta(G_t).$$

summary

computer algebra

- efficient algorithms and software
- for matrices over a field
- for univariate polynomials

Reed-Solomon decoding

- context and unique decoding problem
- key equations and how to solve them
- correcting more errors?

polynomial matrices

- introduction to vector interpolation
- core algorithms & shifted normal forms
- fast divide and conquer interpolation

efficient list decoding

- the Guruswami-Sudan algorithm
- via structured systems or basis reduction
- a word on extensions