Vincent Neiger
LIP6, Sorbonne Université, France

designing and exploiting fast algorithms for univariate polynomial matrices

Journées Nationales de Calcul Formel
Centre International de Rencontre Mathématiques
Marseille Luminy, France, 4 March 2024

outline

- computer algebra
- polynomial matrices
- first algorithms
- exercises

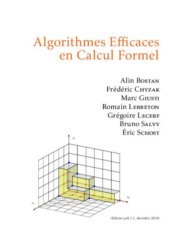
outline

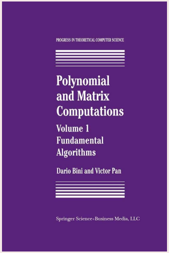
- computer algebra
- polynomial matrices
- first algorithms
- exercises

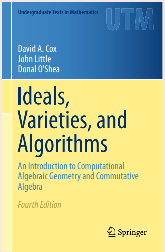
- efficient algorithms and software
- for matrices over a field
- for univariate polynomials

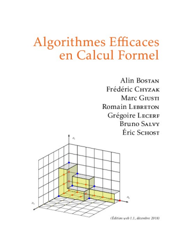

PETER BÜRGISSER
MICHAEL CLAUSEN
M. AMIN SHOKROLLAHI
Grundlehren der mathematischen Wissenschaften
A Series of Comprehensive Studies in Mathematics
ALGEBRAIC COMPLEXITY THEORY
Springer-Verlag Berlin Heidelberg GmbH

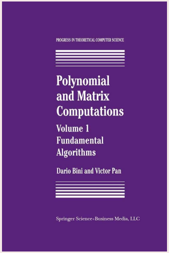

Modern Computer Algebra
Third Edition
Joachim von zur Gathen and Jürgen Gerhard
CAMBRIDGE
CAMBRIDGE
more information – www.cambridge.org/9781107039032

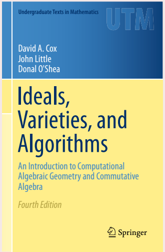

Algorithmes Efficaces en Calcul Formel
Alin Bostan
Frédéric Chyzak
Marc Giusti
Romain Lebreton
Grégoire Lecerf
Bruno Salvy
Éric Schost
(Édition web 1.1, décembre 2018)


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

LinBox
Exact Linear Algebra

GIVARO
Finite Fields

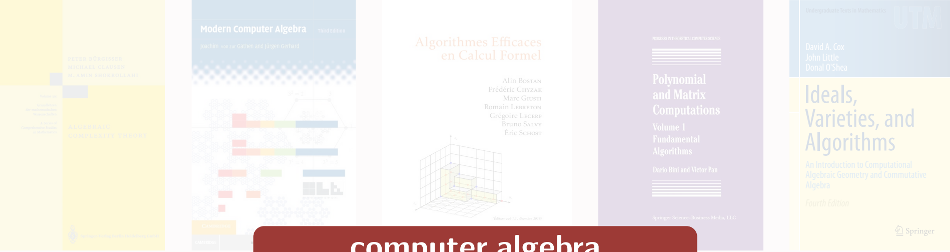
FFLAS-FFPACK
Matrix kernels

GMP
Arbitrary precision Integers

BLAS
Linear Algebra Subroutines

SymPy

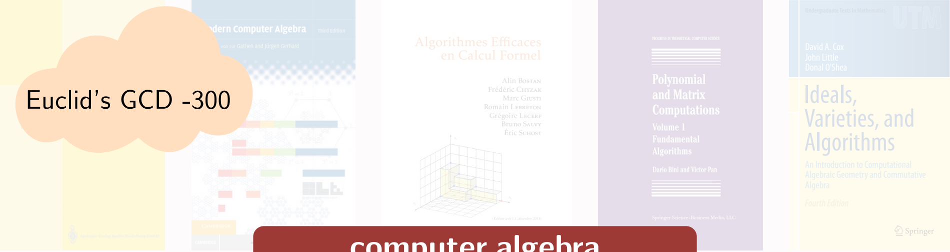
CoCoA



computer algebra

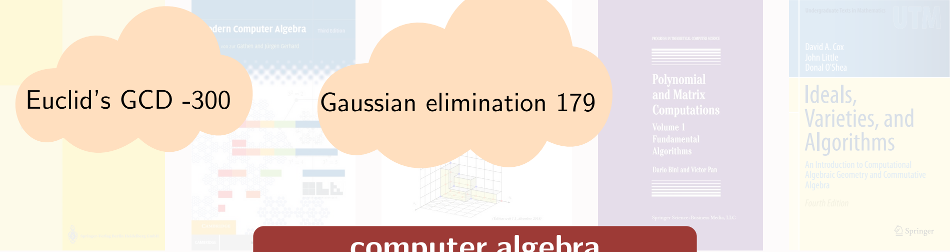
algorithm design
and **software implementations**
for **exact computations**
with **mathematical objects**

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**

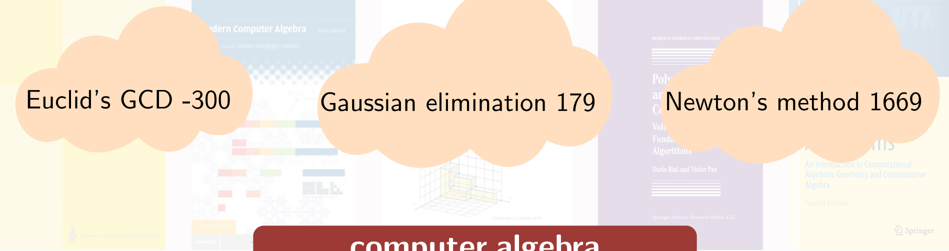
Euclid's GCD -300

Gaussian elimination 179

Newton's method 1669

computer algebra

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



Euclid's GCD -300

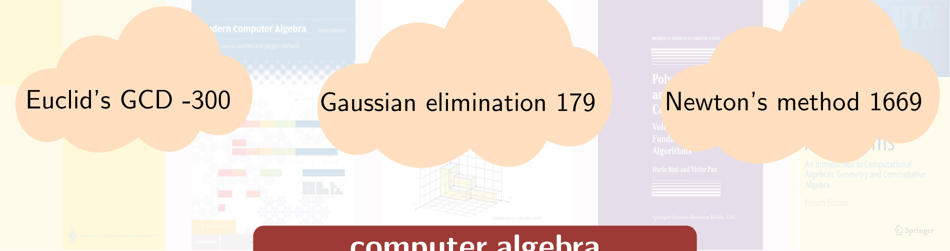
Gaussian elimination 179

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

FFT 1805, '65

Karatsuba '62

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

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

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

Euclid's GCD -300

Gaussian elimination 179

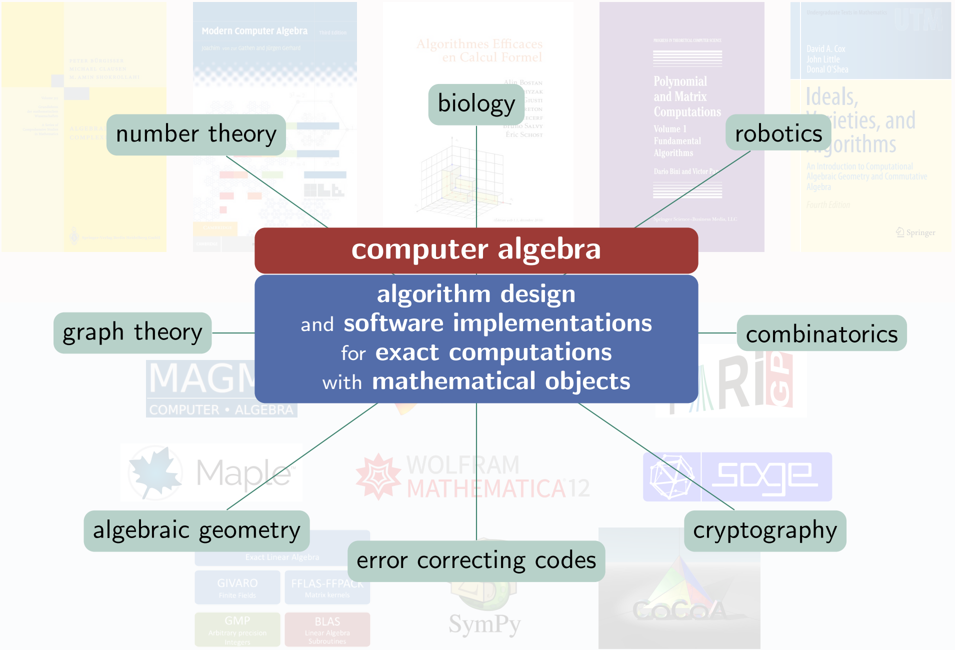
Newton's method 1669

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


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

computer algebra

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

number theory

biology

robotics

graph theory

combinatorics

algebraic geometry

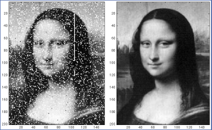
error correcting codes

cryptography

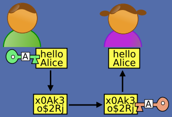
this Friday:

- Alexandre Guillemot: validated numerical methods + interval arithmetic
- Tom Hubrecht: correct rounding of power function

error correcting codes



cryptographic protocols



XXth-XXIst centuries : digital data & interconnected networks

integrity – confidentiality

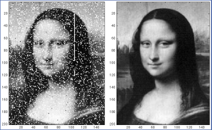
discrete structures : exact and intensive computations

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

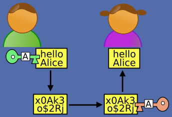
goal of computer algebra

fast algorithms : complexity & efficient implementations

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

general methodology: **reductions** to efficient **basic operations**

- `IntMul`: integer multiplication
- `MatMul`: matrix multiplication
- `PolMul`: univariate 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 $\mathbb{F}_p[x]/\langle f(x) \rangle$
rationals $\mathbb{Q}$, number fields, ...

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 $\mathbb{F}_p[x]/\langle f(x) \rangle$
rationals $\mathbb{Q}$, number fields, ...

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

- 👍 standard complexity model for algebraic computations
- 👍 often well correlated to implementation timings (e.g. over $\mathbb{K} = \mathbb{F}_p$)
- 👎 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 $\mathbb{F}_p[x]/\langle f(x) \rangle$
rationals $\mathbb{Q}$, number fields, ...

practical performance
$\rightsquigarrow$ measure software running time

⚠ strongly influenced by the quality of the implementation

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

$$\mathbf{M} = \begin{bmatrix} 28 & 68 & 75 & 70 \\ 38 & 25 & 75 & 55 \\ 24 & 1 & 56 & 28 \end{bmatrix} \in \mathbb{K}^{3\times 4}$$ $\longrightarrow$ 3×4 matrix over $\mathbb{K}$ (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 $m \gtrsim$ a few 100s
  in NTL, FLINT, fflas-ffpack...

[Coppersmith-Winograd 1990]

- best-known exponent $\omega \approx 2.3719$
  [Le Gall'14] [Alman-Williams'20] [Duan-Wu-Zhou'23]
- "galactic" algorithms: strongly impractical as such

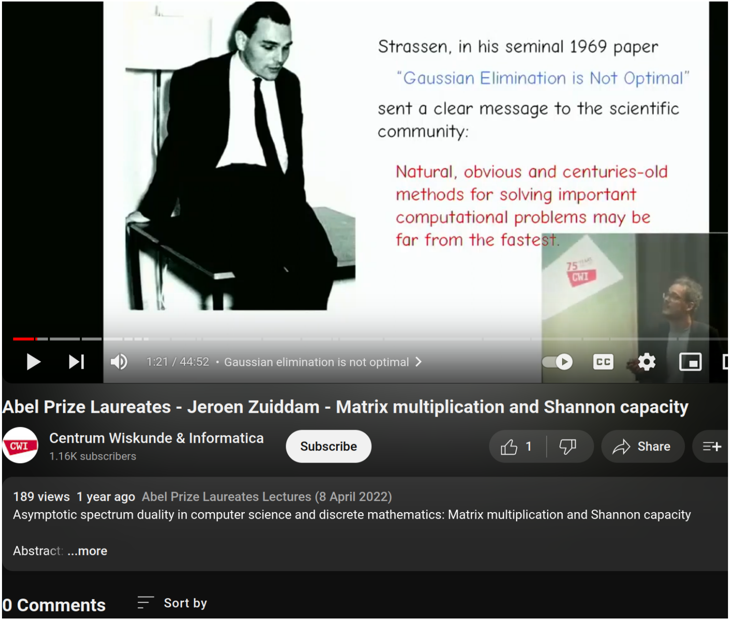
Strassen, in his seminal 1969 paper

"Gaussian Elimination is Not Optimal"

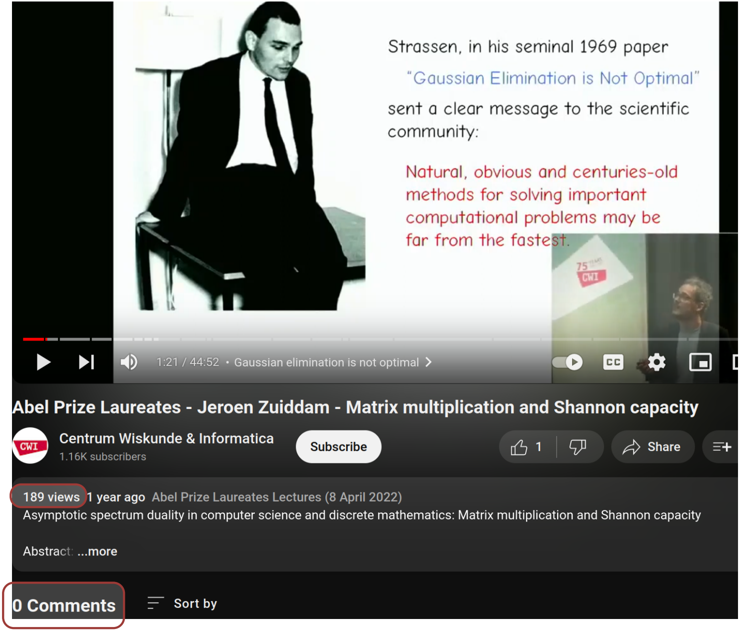
sent a clear message to the scientific community:

Natural, obvious and centuries-old methods for solving important computational problems may be far from the fastest.

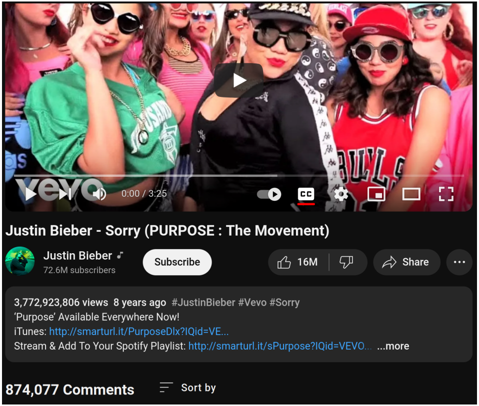
reductions: a new hope

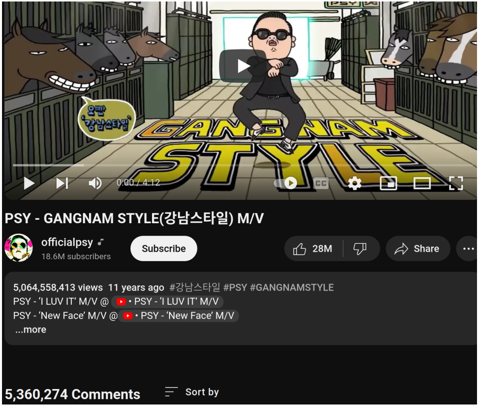


reductions: a new hope

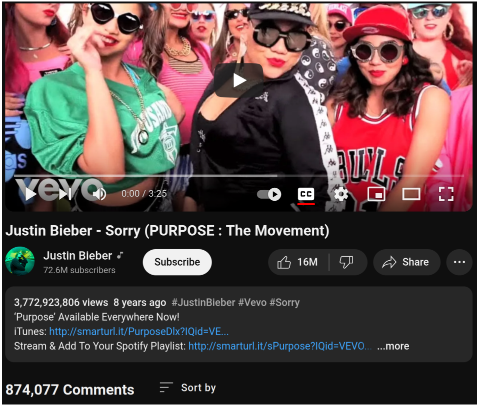


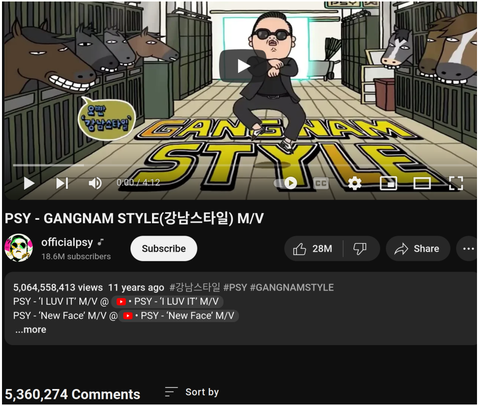
reductions: a new hope





reductions: a new hope





take-home messages:

- bibliometric indicators measure quantity, and there exist counterexamples to "quantity = quality"
- design fast algorithms for the most basic routines $\rightarrow$ MatMul
- design efficient reductions to them for other tasks $\rightarrow$ LinSys, Det, Inverse

polynomials: multiplication

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

$p \in \mathbb{K}[x]_{<8} \quad \longrightarrow$ 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] $\quad M(d) \in O(d^{1.58})$

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

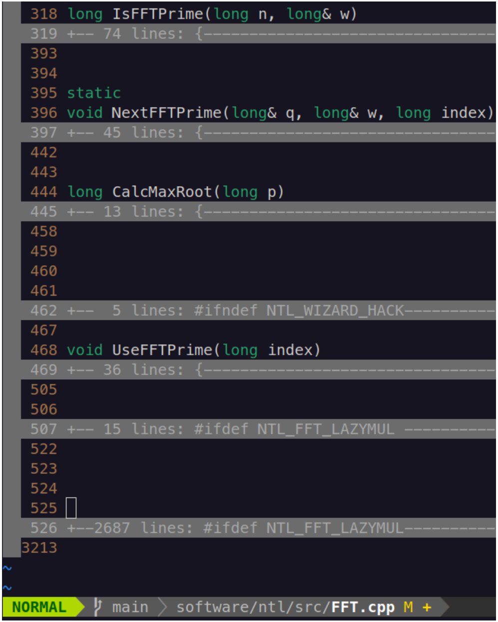
breakthrough: **quasi-linear** polynomial multiplication

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

reductions strike back



- small prime FFT in NTL:

⇝ about **5500 lines** of C++

⇝ target operation: FFT

(including 1200 lines for vectorized version and 1100 for machine word arithmetic. . .)

reductions strike back

```
void DivRem(zz_pX& q, zz_pX& r, const zz_pX& a,
+--- 6 lines: {------------------------------

void div(zz_pX& q, const zz_pX& a, const zz_pX&
+--- 6 lines: {------------------------------

void div(zz_pX& q, const zz_pX& a, zz_p b)
+--- 5 lines: {------------------------------


void rem(zz_pX& r, const zz_pX& a, const zz_pX&
+--- 6 lines: {------------------------------



long operator==(const zz_pX& a, long b)
+--- 20 lines: {-----------------------------

long operator==(const zz_pX& a, zz_p b)
+--- 11 lines: {-----------------------------

void power(zz_pX& x, const zz_pX& a, long e)
+--- 40 lines: {-----------------------------

void reverse(zz_pX& x, const zz_pX& a, long hi)
+--- 13 lines: {-----------------------------

NTL_END_IMPL
~
~
NORMAL  main  software/ntl/src/lzz_pX.cpp
```

- small prime FFT in NTL:

$\rightsquigarrow$ about **5500 lines** of C++

$\rightsquigarrow$ target operation: FFT

(including 1200 lines for vectorized version and 1100 for machine word arithmetic. . .)

- polynomials in $\mathbb{Z}/p\mathbb{Z}[x]$:

$\rightsquigarrow$ about **5500 lines** as well

$\rightsquigarrow$ target operations include:

. multiplication, truncated inversion, division,
. interpolation, multipoint evaluation,
. XGCD, Berlekamp-Massey, resultant,
. power projection, modular composition, . . .

reductions strike back

```
void FFTDiv(zz_pX& q, const zz_pX& a, const zz_pX& b)
{

   long n = deg(b);
   long m = deg(a);
   long k;

+----  4 lines: if (m < n) {------------------------

+----  6 lines: if (m >= 3*n) {---------------------

   zz_pX P1, P2, P3;

   CopyReverse(P3, b, 0, n);
   InvTrunc(P2, P3, m-n+1);
   CopyReverse(P1, P2, 0, m-n);

   k = NextPowerOfTwo(2*(m-n)+1);

   fftRep R1(INIT_SIZE, k), R2(INIT_SIZE, k);

   TofftRep(R1, P1, k);
   TofftRep(R2, a, k, n, m);
   mul(R1, R1, R2);
   FromfftRep(q, R1, m-n, 2*(m-n));
}
```

- small prime FFT in NTL:
  - $\rightsquigarrow$ about **5500 lines** of C++
  - $\rightsquigarrow$ target operation: FFT

(including 1200 lines for vectorized version and 1100 for machine word arithmetic...)

- polynomials in $\mathbb{Z}/p\mathbb{Z}[x]$:
  - $\rightsquigarrow$ about **5500 lines** as well
  - $\rightsquigarrow$ target operations include:
    - . multiplication, truncated inversion, division,
    - . interpolation, multipoint evaluation,
    - . XGCD, Berlekamp-Massey, resultant,
    - . power projection, modular composition, ...

- reductions are often
  - . concise and readable
  - . close to the pseudocode

reductions strike back

```
void FFTDiv(zz_pX& q, const zz_pX& a, const zz_pX& b)
{

   long n = deg(b);
   long m = deg(a);
   long k;

+----  4 lines: if (m < n) {-----------------------------

+----  6 lines: if (m >= 3*n) {--------------------------

   zz_pX P1, P2, P3;

   CopyReverse(P3, b, 0, n);
   InvTrunc(P2, P3, m-n+1);
   CopyReverse(P1, P2, 0, m-n);

   k = NextPowerOfTwo(2*(m-n)+1);

   fftRep R1(INIT_SIZE, k), R2(INIT_SIZE, k);

   TofftRep(R1, P1, k);
   TofftRep(R2, a, k, n, m);
   mul(R1, R1, R2);
   FromfftRep(q, R1, m-n, 2*(m-n));
}
```

- $m \leftarrow \deg(A)$ and $n \leftarrow \deg(B)$
- if $m < n$, return $(0, A)$
- set reversals $\tilde{A} \leftarrow x^m\ A(1/x)$ and $\tilde{B} \leftarrow x^n\ B(1/x)$
- find $\tilde{Q} = \tilde{A}/\tilde{B} \mod x^{m-n+1}$ by power series inversion and product
- reverse $\tilde{Q}$ to obtain Q

reductions strike back

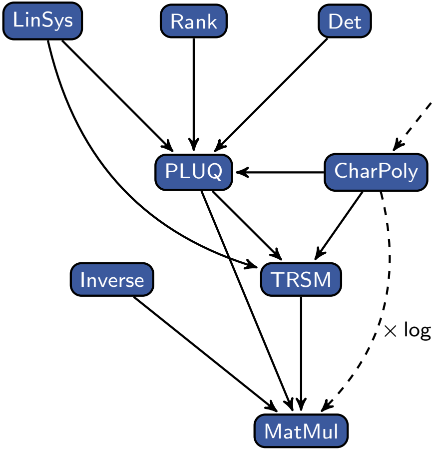
concentrate efforts on: basic routines + good reductions

```
void FFTDiv(zz_pX& q, const zz_pX& a, const zz_pX& b)
{

   long n = deg(b);
   long m = deg(a);
   long k;

+----  4 lines: if (m < n) {-------------------------

+----  6 lines: if (m >= 3*n) {----------------------

   zz_pX P1, P2, P3;

   CopyReverse(P3, b, 0, n);
   InvTrunc(P2, P3, m-n+1);
   CopyReverse(P1, P2, 0, m-n);

   k = NextPowerOfTwo(2*(m-n)+1);

   fftRep R1(INIT_SIZE, k), R2(INIT_SIZE, k);

   TofftRep(R1, P1, k);
   TofftRep(R2, a, k, n, m);
   mul(R1, R1, R2);
   FromfftRep(q, R1, m-n, 2*(m-n));
}
```

- $m \leftarrow \deg(A)$ and $n \leftarrow \deg(B)$
- if $m < n$, return $(0, A)$
- set reversals $\tilde{A} \leftarrow x^m \, A(1/x)$ and $\tilde{B} \leftarrow x^n \, B(1/x)$
- find $\tilde{Q} = \tilde{A}/\tilde{B} \mod x^{m-n+1}$ by power series inversion and product
- reverse $\tilde{Q}$ to obtain Q

matrices: main computational problems

reductions of most problems to matrix multiplication



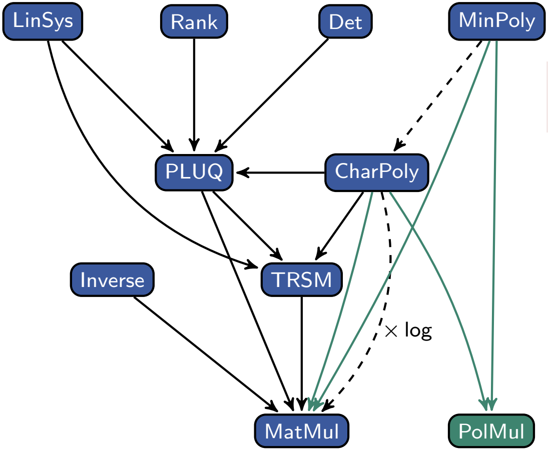
PLUQ = Gaussian elimination
TRSM = triangular solving

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

not closed:
open:

matrices: main computational problems

reductions of most problems to matrix multiplication



PLUQ = Gaussian elimination
TRSM = triangular solving

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

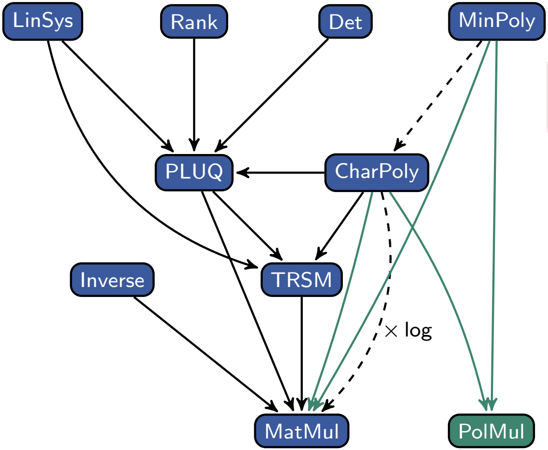
$$\left.\begin{array}{c}\text{MinPoly}\\ \text{CharPoly}\end{array}\right\} = O(\text{MatMul})$$

exploiting **non-naive** PolMul

not closed:
open:

matrices: main computational problems

reductions of most problems to matrix multiplication



PLUQ = Gaussian elimination
TRSM = triangular solving

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

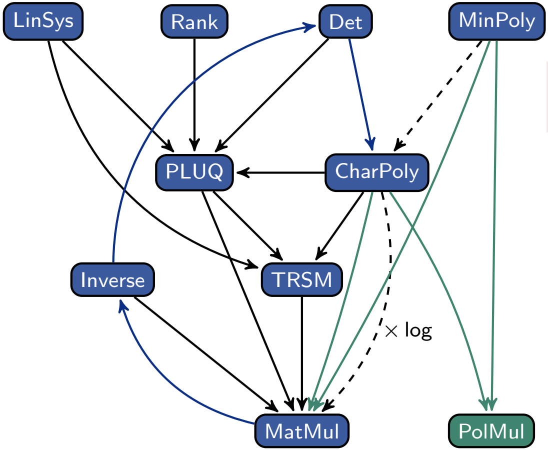
$$\left.\begin{array}{c}\text{MinPoly}\\ \text{CharPoly}\end{array}\right\} = O(\text{MatMul})$$

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

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

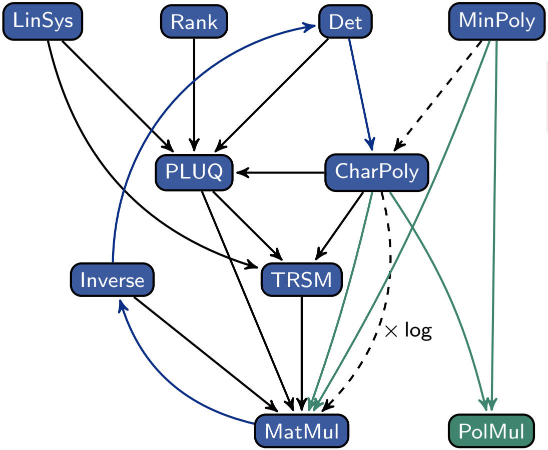
$$\left.\begin{array}{c}\text{MinPoly}\\ \text{CharPoly}\end{array}\right\} = O(\text{MatMul})$$

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

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

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

exploiting **non-naive** PolMul

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

bonus: some notes/references

[Jeannerod-Pernet-Storjohann 2013] `doi.org/10.1016/j.jsc.2013.04.004`

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

"not closed": it is open, but

- there is a randomized algorithm for Frobenius form computation which has complexity $O(\mathrm{MatMul})$

  [Pernet-Storjohann 2007] `http://www.cs.uwaterloo.ca/~astorjoh/cpoly.pdf`
- recent developments give new insight concerning core operations typically used in Frobenius form algorithms

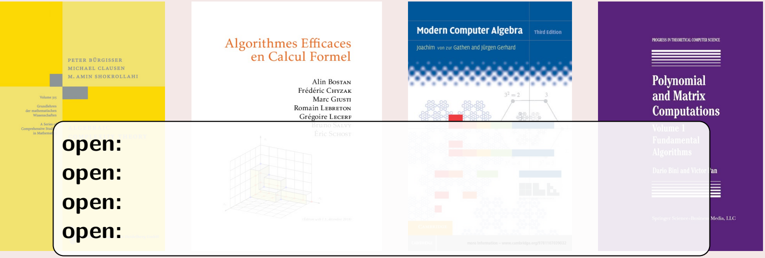
  charpoly in $O(\mathrm{MatMul})$: [Neiger-Pernet 2021] `doi.org/10.1016/S0885-064X(22)00005-X`
  Krylov iterates in $O(\mathrm{MatMul})$: [Neiger-Pernet-Villard 2024] `hal.science/hal-04445355`

polynomials: main computational problems

most problems have quasi-linear complexity

thanks to reductions to `PolMul` — did we mention the importance of good reductions?

- 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



open:

open:

open:

open:

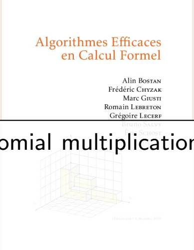
polynomials: main computational problems

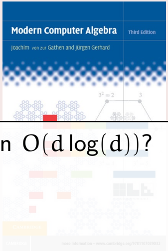
most problems have quasi-linear complexity

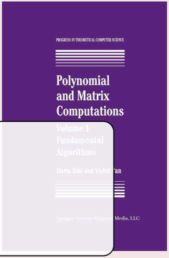
thanks to reductions to `PolMul` — did we mention the importance of good reductions?

$O(M(d))$

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







open: polynomial multiplication in $O(d\log(d))$?
open:
open:
open:

polynomials: main computational problems

most problems have quasi-linear complexity

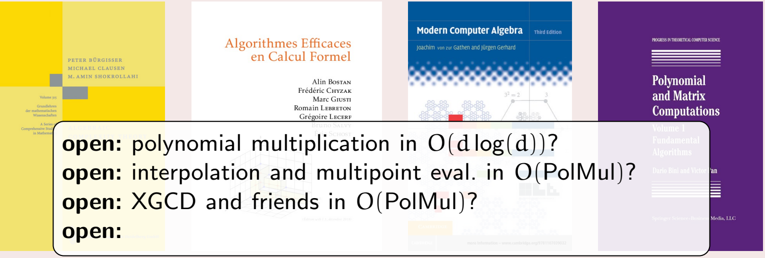
thanks to reductions to `PolMul` — did we mention the importance of good reductions?

$O(M(d))$

- 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



open: polynomial multiplication in $O(d\log(d))$?
open: 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

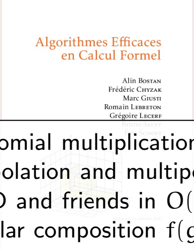
thanks to reductions to `PolMul` — did we mention the importance of good reductions?

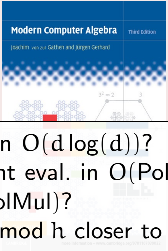
$O(M(d))$

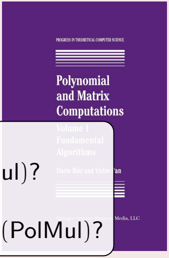
- addition $f + g$, multiplication $f * g$
- division with remainder $f = qg + r$
- truncated inverse $f^{-1} \bmod x^d$

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

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







open: polynomial multiplication in $O(d\log(d))$?
open: 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/references

polynomial multiplication in $O(d\log(d))$?

- 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
  [Harvey-van der Hoeven 2019] `https://doi.org/10.1016/j.jco.2019.03.004`
  [Harvey-van der Hoeven 2022] `https://doi.org/10.1145/3505584`

interpolation and multipoint evaluation in $O(\mathsf{PolMul})$?

- remains open for an arbitrary set of points, with no assumption, but:
- by design, solved for FFT points, when available
- more generally, solved for points forming a geometric sequence
  [Bostan-Schost 2005] `https://doi.org/10.1016/j.jco.2004.09.009`
- in many applications of interpolation/evaluation, one can choose the points, in which case $O(\mathsf{PolMul})$ is feasible

```
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:

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.b
);

        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.

```
            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

SageMath — *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 | with NTL

- `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

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 — with NTL

`PLUQ`	$m = 3800$	1.00s	`PolMul`	$d = 7 \times 10^6$	1.03s
`LinSys`	$m = 3800$	1.00s	`Division`	$d = 4 \times 10^6$	0.96s
`MatMul`	$m = 3000$	0.97s	`XGCD`	$d = 2 \times 10^5$	0.99s
`Inverse`	$m = 2800$	1.01s	`MinPoly`	$d = 2 \times 10^5$	1.10s
`CharPoly`	$m = 2000$	1.09s	`MPeval`	$d = 1 \times 10^4$	1.01s

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
- $\boxed{\texttt{MPeval} \quad d = 1 \times 10^4}$ 1.01s

constant matrices accelerated by polynomial matrices

matrix exponentiation

input: matrix $A \in \mathbb{K}^{m \times m}$,
integer $k > 0$

output: A^k

constant matrices accelerated by polynomial matrices

matrix exponentiation

input: matrix $A \in \mathbb{K}^{m \times m}$,
integer $k > 0$

output: A^k

- repeated squaring: $O(m^\omega \log(k))$

constant matrices accelerated by polynomial matrices

matrix exponentiation

input: matrix $A \in \mathbb{K}^{m \times m}$,
integer $k > 0$

output: A^k

- repeated squaring: $O(m^\omega \log(k))$
- using Frobenius form:
$O(m^\omega \log(m) \log\log(m))$ if $\log(k) \in O(m)$
[Giesbrecht 1995] [Storjohann 2001]
- improvement with polynomial matrices:
$O(m^\omega \log\log(m)^2)$ if $\log(k) \in O(m)$
[Giesbrecht 1995] [Neiger-Pernet-Villard 2024]

constant matrices accelerated by polynomial matrices

matrix exponentiation

input: matrix $A \in \mathbb{K}^{m \times m}$,
integer $k > 0$

output: A^k

can we reach $O(m^\omega)$?

- repeated squaring: $O(m^\omega \log(k))$
- using Frobenius form:
$O(m^\omega \log(m) \log\log(m))$ if $\log(k) \in O(m)$
[Giesbrecht 1995] [Storjohann 2001]
- improvement with polynomial matrices:
$O(m^\omega \log\log(m)^2)$ if $\log(k) \in O(m)$
[Giesbrecht 1995] [Neiger-Pernet-Villard 2024]

constant matrices accelerated by polynomial matrices

matrix exponentiation

input: matrix $A \in \mathbb{K}^{m \times m}$,
integer $k > 0$

output: A^k

can we reach $O(m^\omega)$?

- repeated squaring: $O(m^\omega \log(k))$
- using Frobenius form:
  $O(m^\omega \log(m) \log\log(m))$ if $\log(k) \in O(m)$
  [Giesbrecht 1995] [Storjohann 2001]
- improvement with polynomial matrices:
  $O(m^\omega \log\log(m)^2)$ if $\log(k) \in O(m)$
  [Giesbrecht 1995] [Neiger-Pernet-Villard 2024]

Krylov iterates

input: matrix $A \in \mathbb{K}^{m \times m}$,
vector $v \in \mathbb{K}^{m \times 1}$

output: $v, Av, \dots, A^{m-1}v$

constant matrices accelerated by polynomial matrices

matrix exponentiation

input: matrix $A \in \mathbb{K}^{m \times m}$,
integer $k > 0$

output: A^k

can we reach $O(m^\omega)$?

- repeated squaring: $O(m^\omega \log(k))$
- using Frobenius form:
  $O(m^\omega \log(m) \log\log(m))$ if $\log(k) \in O(m)$
  [Giesbrecht 1995] [Storjohann 2001]
- improvement with polynomial matrices:
  $O(m^\omega \log\log(m)^2)$ if $\log(k) \in O(m)$
  [Giesbrecht 1995] [Neiger-Pernet-Villard 2024]

Krylov iterates

input: matrix $A \in \mathbb{K}^{m \times m}$,
vector $v \in \mathbb{K}^{m \times 1}$

output: $v, Av, \ldots, A^{m-1}v$

- repeated matrix-vector products: $O(m^3)$
- via repeated squaring: $O(m^\omega \log(m))$
  [Keller-Gehrig 1985]

constant matrices accelerated by polynomial matrices

matrix exponentiation

input: matrix $A \in \mathbb{K}^{m \times m}$,
integer $k > 0$

output: A^k

can we reach $O(m^\omega)$?

- repeated squaring: $O(m^\omega \log(k))$
- using Frobenius form:
  $O(m^\omega \log(m) \log\log(m))$ if $\log(k) \in O(m)$
  [Giesbrecht 1995] [Storjohann 2001]
- improvement with polynomial matrices:
  $O(m^\omega \log\log(m)^2)$ if $\log(k) \in O(m)$
  [Giesbrecht 1995] [Neiger-Pernet-Villard 2024]

Krylov iterates

input: matrix $A \in \mathbb{K}^{m \times m}$,
vector $v \in \mathbb{K}^{m \times 1}$

output: $v, Av, \ldots, A^{m-1}v$

- repeated matrix-vector products: $O(m^3)$
- via repeated squaring: $O(m^\omega \log(m))$
  [Keller-Gehrig 1985]
- with polynomial matrices: $O(m^\omega)$
  [Zhou-Labahn-Storjohann 2012][Neiger-Pernet 2021]

constant matrices accelerated by polynomial matrices

matrix exponentiation

input: matrix $A \in \mathbb{K}^{m \times m}$,
integer $k > 0$

output: A^k

can we reach $O(m^\omega)$?

- repeated squaring: $O(m^\omega \log(k))$
- using Frobenius form:
  $O(m^\omega \log(m) \log\log(m))$ if $\log(k) \in O(m)$
  [Giesbrecht 1995] [Storjohann 2001]
- improvement with polynomial matrices:
  $O(m^\omega \log\log(m)^2)$ if $\log(k) \in O(m)$
  [Giesbrecht 1995] [Neiger-Pernet-Villard 2024]

Krylov iterates

input: matrix $A \in \mathbb{K}^{m \times m}$,
vector $v \in \mathbb{K}^{m \times 1}$

output: $v, Av, \ldots, A^{m-1}v$

we do reach $O(m^\omega)$!

... can we do better?

- repeated matrix-vector products: $O(m^3)$
- via repeated squaring: $O(m^\omega \log(m))$
  [Keller-Gehrig 1985]
- with polynomial matrices: $O(m^\omega)$
  [Zhou-Labahn-Storjohann 2012][Neiger-Pernet 2021]

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

sage Python/Cython

goals: **complete, robust, available**

(more than 60k downloads per month)

software development for polynomial matrices

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

Return the ... matrix.

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

INPUT:

... deleted

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

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**

algorithms, vectorization, multithreading

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**

algorithms, vectorization, multithreading

software development for polynomial matrices

Polynomial Matrix Library   *C/C++*

533 files, 72k lines of code, including 21k lines of comments

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

[Hyun-Neiger-Schost '19]

- most tools are 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 compute this determinant with PML?"

wide range of algorithms

efficiency = state of the art

system solving, determinant, kernel,
Hermite-Padé approximation,
high-order lifting, basis reduction...

outline

- computer algebra
- polynomial matrices
- first algorithms
- exercises

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

outline

- **computer algebra**
  - efficient algorithms and software
  - for matrices over a field
  - for univariate polynomials
- **polynomial matrices**
  - basic definitions and properties
  - use in various situations
  - seen as matrices / seen as polynomials
- **first algorithms**
- **exercises**

polynomial matrices

basic definitions and properties

$\mathbb{K}[x]^{m\times n}$ = set of $m \times n$ matrices over $\mathbb{K}[x]$

called polynomial matrices in what follows

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

- structure: matrices over $\mathbb{K}[x]$ $\longleftrightarrow$ free modules over $\mathbb{K}[x]$

similarly to: matrices over $\mathbb{K}$ $\longleftrightarrow$ vector spaces over $\mathbb{K}$

- basic operations: addition and multiplication

defined as usual (multiplication requires compatible dimensions)

- $\mathbb{K}[x]$ is not a field

$\mathbb{K}[x]^{m\times n}$ = set of $m \times n$ matrices over $\mathbb{K}[x]$

called polynomial matrices in what follows

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

- structure: matrices over $\mathbb{K}[x]$ $\longleftrightarrow$ free modules over $\mathbb{K}[x]$
similarly to: matrices over $\mathbb{K}$ $\longleftrightarrow$ vector spaces over $\mathbb{K}$
- basic operations: addition and multiplication
defined as usual (multiplication requires compatible dimensions)
- $\mathbb{K}[x]$ is not a field

$\rightsquigarrow$ algorithms may work in $\mathbb{K}(x)^{m\times n}$, but be careful with "degree explosion"!

large matrices with small degrees:
characteristic polynomial $\det(x\mathbf{I}_m - \mathbf{M}) \in \mathbb{K}[x]$ of a matrix $\mathbf{M} \in \mathbb{K}^{m \times m}$
$\rightsquigarrow$ determinant of polynomial matrix $x\mathbf{I}_m - \mathbf{M} \in \mathbb{K}[x]^{m \times m}$

- fastest known algorithm uses this viewpoint [N.-Pernet, 2021]
- gradually transforms $x\mathbf{I}_m - \mathbf{M}$ to smaller matrices with larger degrees

polynomial matrices

examples you already know

large matrices with small degrees:
characteristic polynomial $\det(x\mathbf{I}_m - \mathbf{M}) \in \mathbb{K}[x]$ of a matrix $\mathbf{M} \in \mathbb{K}^{m \times m}$
$\rightsquigarrow$ determinant of polynomial matrix $x\mathbf{I}_m - \mathbf{M} \in \mathbb{K}[x]^{m \times m}$

- fastest known algorithm uses this viewpoint [N.-Pernet, 2021]
- gradually transforms $x\mathbf{I}_m - \mathbf{M}$ to smaller matrices with larger degrees

small matrices with large degree:
extended GCD $uf + vg = \gcd(f, g)$ for polynomials $f, g \in \mathbb{K}[x]_{\leqslant d}$
$\rightsquigarrow$ corresponds to a polynomial matrix transformation

$$\begin{bmatrix} u & v \\ \tilde{g} & \tilde{f} \end{bmatrix} \begin{bmatrix} f \\ g \end{bmatrix} = \begin{bmatrix} \gcd(f, g) \\ 0 \end{bmatrix}$$

with the leftmost (polynomial) matrix of determinant in $\mathbb{K} \setminus \{0\}$

- fastest known “half-gcd” algorithms use this viewpoint

[Knuth, 1970] [Schönhage, 1971] [Brent-Gustavson-Yun, 1980]

polynomial matrices

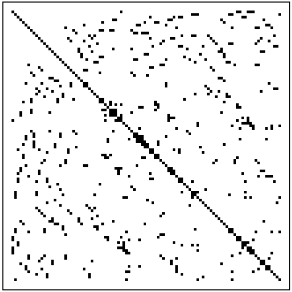
use in various situations

operations on sparse matrices

- solving sparse linear systems over $\mathbb{K}$
- computing the minimal polynomial / Frobenius form
- introducing parallelism in these computations

[Wiedemann 1986]
[Coppersmith 1993]
[Villard 1997]

example of sparse matrix in $\mathbb{K}^{m \times m}$
typical case: $O(m)$ nonzero entries



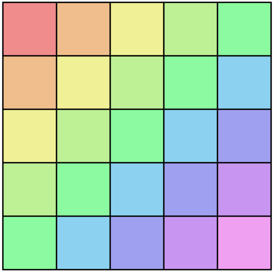
uses **polynomial matrix** generator
of linearly recurrent **matrix** sequence

use in various situations

operations on structured matrices
- matrix-vector multiplication
- linear system solving
- nullspace computation

[Kailath-Kung-Morf 1979]
[Bostan et al. 2017]

example of Hankel matrix
$\rightsquigarrow$ block-Hankel matrices
$\rightsquigarrow$ Hankel-like matrices



uses **polynomial matrix** multiplication and
matrix-Padé approximation / **matrix**-GCD

bivariate interpolation and multipoint evaluation

problem: given points $(\alpha_1, \beta_1), \ldots, (\alpha_n, \beta_n)$ in $\mathbb{K}^2$,

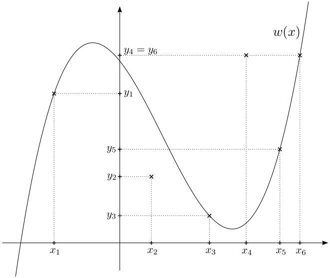
- given $p(x, y)$, compute $p(\alpha_i, \beta_i)$ for $1 \leqslant i \leqslant n$
- find $p(x, y)$ of small degree such that $p(\alpha_i, \beta_i) = 0$

[Nüsken-Ziegler 2004]

[Beckermann 1992] [van Barel-Bultheel 1992]
[Marinari-Möller-Mora 1993]

bivariate interpolation = main step
in Reed-Solomon list-decoding
(univariate interpolation with errors)

[Guruswami-Sudan 1999] [Kötter-Vardy 2003]



uses **polynomial matrix** multiplication and
matrix rational reconstruction / **algebraic approximants**

polynomial matrices

seen as matrices over $\mathbb{K}(x)$

linear algebra viewpoint:

matrices in $\mathbb{K}[x]^{m \times n}$ are also in $\mathbb{K}(x)^{m \times n}$

(and $\mathbb{K}(x)$ is a field)

$\Rightarrow$ usual definition of addition, multiplication, determinant
these do not involve division anyway (... in algorithms?)

$\Rightarrow$ usual definition of rank
coincides with rank of free module

$\Rightarrow$ usual definition of inverse
with inverse over $\mathbb{K}(x)$

polynomial matrices

seen as matrices over $\mathbb{K}(x)$

linear algebra viewpoint:

matrices in $\mathbb{K}[x]^{m\times n}$ are also in $\mathbb{K}(x)^{m\times n}$

(and $\mathbb{K}(x)$ is a field)

$\Rightarrow$ usual definition of addition, multiplication, determinant
these do not involve division anyway (... in algorithms?)

$\Rightarrow$ usual definition of rank
coincides with rank of free module

$\Rightarrow$ usual definition of inverse
with inverse over $\mathbb{K}(x)$

inverse is over $\mathbb{K}[x]$ $\Leftrightarrow$ $\det(\mathbf{A}) \in \mathbb{K}\setminus\{0\}$

def.: A is **unimodular**

polynomial matrices

seen as matrices over $\mathbb{K}(x)$

linear algebra viewpoint:

matrices in $\mathbb{K}[x]^{m\times n}$ are also in $\mathbb{K}(x)^{m\times n}$

(and $\mathbb{K}(x)$ is a field)

$\Rightarrow$ usual definition of addition, multiplication, determinant
these do not involve division anyway (... in algorithms?)

$\Rightarrow$ usual definition of rank
coincides with rank of free module

$\Rightarrow$ usual definition of inverse
with inverse over $\mathbb{K}(x)$

$\rightsquigarrow$ algorithms may work in $\mathbb{K}(x)^{m\times n}$, but be careful with "degree explosion"!

exercise: Gaussian elimination is exponential-time

polynomial matrices

seen as matrices over $\mathbb{K}(x)$

linear algebra viewpoint:

matrices in $\mathbb{K}[x]^{m\times n}$ are also in $\mathbb{K}(x)^{m\times n}$

(and $\mathbb{K}(x)$ is a field)

- viewpoint useful for definitions and properties
- viewpoint hardly usable for algorithms:
  ignores degree growth + too coarse cost bounds

. cost of naive addition in $\mathbb{K}[x]^{m\times n}$ $\longrightarrow$ $O(mn)$ additions in $\mathbb{K}(x)$

. cost of naive multiplication in $\mathbb{K}[x]^{m\times m}$ $\longrightarrow$ $O(m^3)$ ops in $\mathbb{K}(x)$

polynomial matrices

seen as matrices over $\mathbb{K}(x)$

linear algebra viewpoint:

matrices in $\mathbb{K}[x]^{m\times n}$ are also in $\mathbb{K}(x)^{m\times n}$

(and $\mathbb{K}(x)$ is a field)

- viewpoint useful for definitions and properties
- viewpoint hardly usable for algorithms:
  ignores degree growth + too coarse cost bounds

. cost of naive addition in $\mathbb{K}[x]^{m\times n}$ $\longrightarrow$ $O(mn)$ additions in $\mathbb{K}(x)$

. cost of naive multiplication in $\mathbb{K}[x]^{m\times m}$ $\longrightarrow$ $O(m^3)$ ops in $\mathbb{K}(x)$

for algorithms&complexity, considering the degrees of entries is essential

polynomial matrices

seen as polynomials over $\mathbb{K}^{m\times n}$

polynomial viewpoint:

$\mathbb{K}[x]^{m\times n}$ **is isomorphic to** $\mathbb{K}^{m\times n}[x]$

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

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

polynomial matrices

seen as polynomials over $\mathbb{K}^{m\times n}$

polynomial viewpoint:

$\mathbb{K}[x]^{m\times n}$ is isomorphic to $\mathbb{K}^{m\times n}[x]$

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

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

$\mathbf{A}$ has degree 3; in general, $\deg(\mathbf{AB}) \leqslant \deg(\mathbf{A}) + \deg(\mathbf{B})$
e.g. $\deg(\mathbf{A}^2) = 6$, and $\deg(\mathbf{A}^3) = 8$, and $\deg(\mathbf{A}^4) = 11$

polynomial matrices

seen as polynomials over $\mathbb{K}^{m \times n}$

polynomial viewpoint:

$\mathbb{K}[x]^{m \times n}$ **is isomorphic to** $\mathbb{K}^{m \times n}[x]$

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

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

degree growth enhances computational aspects

example: computing the N-th power A^N

polynomial matrices

seen as polynomials over $\mathbb{K}^{m\times n}$

polynomial viewpoint:

$\mathbb{K}[x]^{m\times n}$ is isomorphic to $\mathbb{K}^{m\times n}[x]$

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

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

degree growth enhances computational aspects

example: computing the N-th power A^N

repeated squaring:

$$\begin{cases} \mathbf{A} \times \mathbf{A} & (\deg = 3) \\ \mathbf{A}^2 \times \mathbf{A}^2 & (\deg \leqslant 6) \\ \vdots & \vdots \\ \mathbf{A}^{\frac{N}{4}} \times \mathbf{A}^{\frac{N}{4}} & (\deg \leqslant \frac{3N}{4}) \\ \mathbf{A}^{\frac{N}{2}} \times \mathbf{A}^{\frac{N}{2}} & (\deg \leqslant \frac{3N}{2}) \end{cases}$$

find small recurrence + unroll it:
[Flajolet-Salvy 1997][Bostan-Neiger-Yurkevich 2023]

$O(N)$ operations in $\mathbb{K}$

- faster than multiplying $\mathbf{A}^{\frac{N}{2}} \times \mathbf{A}^{\frac{N}{2}}$
- does not require FFT
- prototype: $N = 2^{20} \rightsquigarrow$ 1.6s vs. 11.5s

polynomial matrices

seen as polynomials over $\mathbb{K}^{m\times n}$

polynomial viewpoint:

$\mathbb{K}[x]^{m\times n}$ is isomorphic to $\mathbb{K}^{m\times n}[x]$

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

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

- natural notion of degree of a polynomial matrix
- addition of $\mathbf{A}, \mathbf{B} \in \mathbb{K}[x]^{m\times n}$ is in $O(mnd)$ operations in $\mathbb{K}$ where $d = \min(\deg(\mathbf{A}), \deg(\mathbf{B}))$
- some other polynomial operations available: truncation $\mathbf{A}$ rem x^N, shift $x^d\mathbf{A}$, evaluation $\mathbf{A}(\alpha)$ for $\alpha \in \mathbb{K}$

polynomial matrices

seen as polynomials over $\mathbb{K}^{m\times n}$

polynomial viewpoint:

$\mathbb{K}[x]^{m\times n}$ **is isomorphic to** $\mathbb{K}^{m\times n}[x]$

when $m = n$, $\mathbb{K}^{m\times m}$ is a (non-commutative) ring

derived from univariate polynomial algorithms:

- **multiplication** in $\mathbb{K}[x]^{m\times m}$ seen as a product of polynomials
complexity? $O(m^\omega M(d))$ is tempting... and true for best known $M(d)$

- **truncated inversion** via power series & Newton iteration
condition for invertibility? complexity?

- fast **Euclidean division with remainder**
conditions for feasibility? complexity?

polynomial matrices

seen as polynomials over $\mathbb{K}^{m\times n}$

polynomial viewpoint:

$\mathbb{K}[x]^{m\times n}$ **is isomorphic to** $\mathbb{K}^{m\times n}[x]$

algorithmically fruitful viewpoint, with some limitations

ignores heterogeneous degrees of matrix entries

consider $\mathbf{A} = \begin{bmatrix} f(x) & a_{01} & \cdots \\ a_{10} & a_{11} & \\ \vdots & & \ddots \end{bmatrix} \in \mathbb{K}[x]^{m\times m}$,

$f(x)$ of degree d, other entries in $\mathbb{K}$

- data structure: $d+1$ matrices in $\mathbb{K}^{m\times m}$
- size of representation: $m^2(d+1)$ $\quad \rightarrow m^2 + d?$
- adding two such matrices: $O(m^2(d+1))$ $\quad \rightarrow m^2 + d?$

outline

computer algebra

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

polynomial matrices

- basic definitions and properties
- use in various situations
- seen as matrices / seen as polynomials

first algorithms

exercises

outline

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

- polynomial matrices
  - basic definitions and properties
  - use in various situations
  - seen as matrices / seen as polynomials

- first algorithms
  - exploiting evaluation-interpolation
  - extending algorithms for polynomials
  - partial linearization techniques

- exercises

naive multiplication: $O(m^3 d^2)$ operations in $\mathbb{K}$ $O(m^\omega M(d))$?

naive multiplication: $O(m^3d^2)$ operations in $\mathbb{K}$ $O(m^\omega M(d))$?

On fast multiplication of polynomials over arbitrary algebras

David G. Cantor[1] **and Erich Kaltofen**[2]★

[1] Department of Mathematics, University of California, Los Angeles, CA 90024-1555, USA
[2] Department of Computer Science, Rensselaer Polytechnic Institute, Troy, NY 12180-3590, USA

Received January 22, 1988 / May 10, 1991

1 Introduction

In this paper we generalize the well-known Schönhage-Strassen algorithm for multiplying large integers to an algorithm for multiplying polynomials with coefficients from an arbitrary, not necessarily commutative, not necessarily associative, algebra $\mathscr{A}$. Our main result is an algorithm to multiply polynomials of degree $< n$ in $O(n \log n)$ algebra multiplications and $O(n \log n \operatorname{loglog} n)$ algebra additions/subtractions (we count a subtraction as an addition). The constant implied by the "O" does not depend upon the algebra $\mathscr{A}$. The parallel complexity of our algorithm, i.e., the depth of the corresponding arithmetic circuit, is

naive multiplication: $O(m^3 d^2)$ operations in $\mathbb{K}$ $O(m^\omega M(d))$?

On fast multiplication of polynomials over arbitrary algebras

David G. Cantor[1] and Erich Kaltofen[2]★

[1] Department of Mathematics, University of California, Los Angeles, CA 90024-1555, USA
[2] Department of Computer Science, Rensselaer Polytechnic Institute, Troy, NY 12180-3590, USA

Received January 22, 1988 / May 10, 1991

multiplication in $\mathbb{K}^{m\times m}[x]$ with degree $\leqslant d$:

- $O(d\log(d))$ multiplications in $\mathbb{K}^{m\times m}$
- $O(d\log(d)\log\log(d))$ additions in $\mathbb{K}^{m\times m}$

$MM(m,d) \in O(m^\omega d\log(d) + m^2 d\log(d)\log\log(d))$

1 Introduction

In this paper we generalize the well-known Schönhage-Strassen algorithm for multiplying large integers to an algorithm for multiplying polynomials with coefficients from an arbitrary, not necessarily commutative, not necessarily associative, algebra $\mathscr{A}$. Our main result is an algorithm to multiply polynomials of degree $<n$ in $O(n \log n)$ algebra multiplications and $O(n \log n \operatorname{loglog} n)$ algebra additions/subtractions (we count a subtraction as an addition). The constant implied by the "O" does not depend upon the algebra $\mathscr{A}$. The parallel complexity of our algorithm, i.e., the depth of the corresponding arithmetic circuit, is

exercise: multiplication, determinant, inversion

1. adapting the evaluation-interpolation paradigm to matrices in $\mathbb{K}[x]^{m\times m}$,

- give an explicit **multiplication** algorithm
- give a **determinant** algorithm
- give an **inversion** algorithm

computing the inverse over the fractions $\mathbb{K}(x)$

2. for each of these algorithms,

- give a required lower bound on the **cardinality of** $\mathbb{K}$
- state and prove an upper bound on the **complexity**

hint: use known degree bounds on the output exploiting evaluation-interpolation

exercise: multiplication, determinant, inversion

1. adapting the evaluation-interpolation paradigm to matrices in $\mathbb{K}[x]^{m\times m}$,

- give an explicit **multiplication** algorithm
- give a **determinant** algorithm
- give an **inversion** algorithm

computing the inverse over the fractions $\mathbb{K}(x)$

2. for each of these algorithms,

- give a required lower bound on the **cardinality of** $\mathbb{K}$
- state and prove an upper bound on the **complexity**

multiplication: for large enough $\mathbb{K}$,
$\mathsf{MM}(m, d) \in O(m^\omega d + m^2 \mathsf{M}(d))$ [Bostan-Schost 2005]

$\rightsquigarrow$ better than $m^\omega \mathsf{M}(d)$

first algorithms

exploiting evaluation-interpolation

exercise: multiplication, determinant, inversion

1. adapting the evaluation-interpolation paradigm to matrices in $\mathbb{K}[x]^{m \times m}$,

- give an explicit **multiplication** algorithm
- give a **determinant** algorithm
- give an **inversion** algorithm

computing the inverse over the fractions $\mathbb{K}(x)$

2. for each of these algorithms,

- give a required lower bound on the **cardinality of** $\mathbb{K}$
- state and prove an upper bound on the **complexity**

	evaluation-interpolation, large $\mathbb{K}$	best known, unconditional
determinant	$O\tilde{}(m^{\omega+1} d)$	$O\tilde{}(m^{\omega} d)$
inversion	$O\tilde{}(m^{\omega+1} d)$	$O\tilde{}(m^{3} d)$
	reductions to `PolMul&MatMul`	reductions to `PolMatMul`

extending algorithms for polynomials

truncated inversion — from book “AECF”

Entrée Un entier $N > 0$, $F \bmod X^N$ une série tronquée.
Sortie $F^{-1} \bmod X^N$.
Si $N = 1$, alors renvoyer f_0^{-1}, où $f_0 = F(0)$.
Sinon :
1. Calculer récursivement l'inverse G de F mod $X^{\lceil N/2 \rceil}$.
2. Renvoyer $G + (1 - GF)G \bmod X^N$.

Algorithme 3.2 – Inverse de série par itération de Newton.

Convergence quadratique pour l'inverse d'une série formelle

Lemme 3.2 Soient $\mathbb{A}$ un anneau non nécessairement commutatif, $F \in \mathbb{A}[[X]]$ une série formelle de terme constant inversible et G une série telle que $G - F^{-1} = O(X^n)$ ($n \geq 1$). Alors la série

$$\mathcal{N}(G) = G + (1 - GF)G \tag{3.2}$$

vérifie $\mathcal{N}(G) - F^{-1} = O(X^{2n})$.

truncated inversion — results

consider a (square) polynomial matrix $\mathbf{A} \in \mathbb{K}[x]^{m \times m}$

- $\mathbf{A}$ is invertible as a power series
  $\Leftrightarrow$ its constant term $\mathbf{A}(0) \in \mathbb{K}^{m \times m}$ is invertible
- if $\mathbf{A}$ is invertible as a power series,
computing $\mathbf{A}^{-1} \bmod x^N$ costs $O(\mathsf{MM}(m, N))$ operations in $\mathbb{K}$

- no additional log: $\mathsf{MM}(m, \frac{N}{2}) + \mathsf{MM}(m, \frac{N}{4}) + \mathsf{MM}(m, \frac{N}{8}) + \cdots$
- excellent reduction to `PolMatMul`!
- timings with the Polynomial Matrix Library:

m	d	`PolMatMul`	`TruncInv`
10	20000	0.203	0.551
20	5000	0.225	0.639
40	2500	0.528	1.424
80	1250	1.227	3.653

division with remainder

problem:
given $\mathbf{A}, \mathbf{B} \in \mathbb{K}^{m \times m}[x]$,
compute $\mathbf{Q}, \mathbf{R} \in \mathbb{K}^{m \times m}[x]$ such that

$$\mathbf{A} = \mathbf{B}\mathbf{Q} + \mathbf{R} \quad \text{and} \quad \deg(\mathbf{R}) < \deg(\mathbf{B})$$

. . . are we not missing an assumption?

first algorithms

extending algorithms for polynomials

division with remainder

problem:
given $\mathbf{A}, \mathbf{B} \in \mathbb{K}^{m \times m}[x]$,
compute $\mathbf{Q}, \mathbf{R} \in \mathbb{K}^{m \times m}[x]$ such that

$$\mathbf{A} = \mathbf{B}\mathbf{Q} + \mathbf{R} \quad \text{and} \quad \deg(\mathbf{R}) < \deg(\mathbf{B})$$

... are we not missing an assumption?

rule 1: dividing by zero is generally a bad idea
rule 2: if you think you need to divide by zero, refer to rule 1
rule 3: neglecting to check that something is not zero does not make it nonzero
etc. etc.

extending algorithms for polynomials

division with remainder

problem:
given $\mathbf{A}, \mathbf{B} \in \mathbb{K}^{m \times m}[x]$,
compute $\mathbf{Q}, \mathbf{R} \in \mathbb{K}^{m \times m}[x]$ such that

$$\mathbf{A} = \mathbf{B}\mathbf{Q} + \mathbf{R} \quad \text{and} \quad \deg(\mathbf{R}) < \deg(\mathbf{B})$$

... are we not missing an assumption?

for a polynomial $p \in \mathcal{A}[x]$, over some ring $\mathcal{A}$, division by p is feasible

- if p is monic (leading coefficient $1_{\mathcal{A}}$)
- and more generally if the leading coefficient of p is invertible in $\mathcal{A}$

assumption: the leading coefficient of $\mathbf{B}$ is invertible in $\mathbb{K}^{m \times m}$

recall $B = B_0 + B_1 x + \cdots + B_d x^d$ with $B_i \in \mathbb{K}^{m \times m}$

division with remainder

problem:
given $\mathbf{A}, \mathbf{B} \in \mathbb{K}^{m \times m}[x]$ with lc$(\mathbf{B})$ invertible,
compute $\mathbf{Q}, \mathbf{R} \in \mathbb{K}^{m \times m}[x]$ such that

$$\mathbf{A} = \mathbf{B}\mathbf{Q} + \mathbf{R} \quad \text{and} \quad \deg(\mathbf{R}) < \deg(\mathbf{B})$$

- under this assumption, the usual fast Euclidean algorithm works
- recall:
  1. reverse the equation,
  2. compute quotient by truncated inverse multiplication
  $$\tilde{\mathbf{Q}} = \tilde{\mathbf{B}}^{-1}\tilde{\mathbf{A}} \mod x^{d_{\mathbf{A}} - d_{\mathbf{B}} + 1}$$
  3. deduce remainder
- complexity is $O(\underbrace{\mathsf{MM}(m, d_{\mathbf{A}} - d_{\mathbf{B}})}_{\text{find } Q} + \underbrace{\mathsf{MM}(m, d_{\mathbf{B}})}_{\text{find } R})$

first algorithms

extending algorithms for polynomials

division with remainder

problem:
given $\mathbf{A}, \mathbf{B} \in \mathbb{K}^{m \times m}[x]$ with $\mathrm{lc}(\mathbf{B})$ invertible,
compute $\mathbf{Q}, \mathbf{R} \in \mathbb{K}^{m \times m}[x]$ such that

$$\mathbf{A} = \mathbf{B}\mathbf{Q} + \mathbf{R} \quad \text{and} \quad \deg(\mathbf{R}) < \deg(\mathbf{B})$$

m	$d_{\mathbf{A}}$	$d_{\mathbf{B}}$	`PolMatMul` in deg $d_{\mathbf{B}}$	`TruncInv` in deg $d_{\mathbf{B}}$	`QuoRem`
10	40000	20000	0.203	0.551	1.873
20	10000	5000	0.225	0.639	2.164
40	5000	2500	0.528	1.424	6.468
80	2500	1250	1.227	3.653	15.59

first algorithms

extending algorithms for polynomials

division with remainder

problem:
given $\mathbf{A}, \mathbf{B} \in \mathbb{K}^{m \times m}[x]$ with $\mathrm{lc}(\mathbf{B})$ invertible,
compute $\mathbf{Q}, \mathbf{R} \in \mathbb{K}^{m \times m}[x]$ such that
$$\mathbf{A} = \mathbf{B}\mathbf{Q} + \mathbf{R} \quad \text{and} \quad \deg(\mathbf{R}) < \deg(\mathbf{B})$$

```
    // step 1: reverse input matrices
    row_reverse(Brev, B, rdegB);
    row_reverse(buf, A, rdegA);

    // step 2: compute quotient
    // Qrev = Brev^{-1} R mod X^{d+1}
    solve_series(Qrev, Brev, buf, d+1);
    reverse(Q, Qrev, d);

    // step 3: deduce remainder
    // R = A - B*Q
    multiply(buf, B, Q);
    sub(R, A, buf);
```

- an efficient reduction, again

- `rdegA` vs. degree d?
- `row_reverse` vs. `reverse` ?

- refinement of matrix degree:

row- or column-wise degrees
$\rightsquigarrow$ improves applicability & complexity

e.g. division by $\mathbf{B} = \mathrm{diag}(x^{d_1}, \ldots, x^{d_m})$

row degree of a polynomial matrix
= the list of the maximum degree in each of its rows

for $\mathbf{A} = (a_{i,j}) \in \mathbb{K}[x]^{m \times n}$,

$$\begin{aligned} \mathrm{rdeg}(\mathbf{A}) &= (\mathrm{rdeg}(\mathbf{A}_{1,*}), \ldots, \mathrm{rdeg}(\mathbf{A}_{m,*})) \\ &= \left(\max_{1 \leqslant j \leqslant n} \deg(\mathbf{A}_{1,j}), \; \ldots, \; \max_{1 \leqslant j \leqslant n} \deg(\mathbf{A}_{m,j}) \right) \in \mathbb{Z}^m \end{aligned}$$

row degree of a polynomial matrix
$=$ the list of the maximum degree in each of its rows

column degree of a polynomial matrix
$=$ the list of the maximum degree in each of its columns

first algorithms

refined degree measures — generalized division

row degree of a polynomial matrix
= the list of the maximum degree in each of its rows

column degree of a polynomial matrix
= the list of the maximum degree in each of its columns

sum of degrees of all entries $\leqslant$ $\begin{matrix} n\times \text{ sum of row degrees} \\ m\times \text{ sum of column degrees} \end{matrix}$ $\leqslant mn\times$ global degree

with notation:

$$\sum_{i,j} \deg(a_{ij}) \leqslant \begin{matrix} n|\mathrm{rdeg}(\mathbf{A})| \\ m|\mathrm{cdeg}(\mathbf{A})| \end{matrix} \leqslant mn \deg(\mathbf{A})$$

first algorithms

refined degree measures — generalized division

row degree of a polynomial matrix
= the list of the maximum degree in each of its rows

column degree of a polynomial matrix
= the list of the maximum degree in each of its columns

sum of degrees of all entries $\leqslant$ $\begin{array}{c} n\times \text{ sum of row degrees} \\ m\times \text{ sum of column degrees} \end{array}$ $\leqslant mn\times$ global degree

with notation:

$$\sum_{i,j} \deg(a_{ij}) \leqslant \begin{array}{c} n|\mathrm{rdeg}(\mathbf{A})| \\ m|\mathrm{cdeg}(\mathbf{A})| \end{array} \leqslant mn \deg(\mathbf{A})$$

consider $\mathbf{A}$ with degree matrix

$$\begin{pmatrix} 100 & 5 & 20 & 1 \\ 100 & 5 & 20 & 1 \\ 100 & 5 & 20 & 1 \\ 100 & 5 & 20 & 1 \end{pmatrix}$$

determinant of $\mathbf{A}$: degree $\leqslant 126$

$\rightsquigarrow$ better than naive bound $4\deg(\mathbf{A}) = 400$

first algorithms

refined degree measures — generalized division

row degree of a polynomial matrix
= the list of the maximum degree in each of its rows

column degree of a polynomial matrix
= the list of the maximum degree in each of its columns

sum of degrees of all entries $\leqslant \begin{array}{c} n\times \text{ sum of row degrees} \\ m\times \text{ sum of column degrees} \end{array} \leqslant mn\times$ global degree

with notation:

$$\sum_{i,j} \deg(a_{ij}) \leqslant \begin{array}{c} n|\mathrm{rdeg}(\mathbf{A})| \\ m|\mathrm{cdeg}(\mathbf{A})| \end{array} \leqslant mn \deg(\mathbf{A})$$

more general division with remainder:

- take for $\mathrm{lc}(\mathbf{B})$ **row-wise** leading coefficients
- if $\mathrm{lc}(\mathbf{B})$ is invertible, division by $\mathbf{B}$ is feasible
- with **row-wise** degree bounds on remainder

$$\begin{bmatrix} 4x^3+2x+2 & 6x^3+2x^2+5 \\ 4x^2+2 & 3x^3+x+3 \end{bmatrix} = \begin{bmatrix} x & 0 \\ 0 & 2x^2 \end{bmatrix} \mathbf{Q} + \begin{bmatrix} 2 & 5 \\ 2 & x+3 \end{bmatrix}$$

reduce **unbalanced** degrees to some **average** degree

where degree means row degree, column degree, or related refined measures

[Storjohann 2006] [Zhou-Labahn 2012] [Jeannerod-Neiger-Villard 2020]

typical properties:

from a matrix $\mathbf{A} \in \mathbb{K}[x]^{m \times m}$ with $D = |\mathrm{rdeg}(\mathbf{A})| \ll m \deg(\mathbf{A})$
construct a matrix $\bar{\mathbf{A}} \in \mathbb{K}[x]^{m' \times m'}$ with

- a slight increase of matrix dimension: $m \leqslant m' \leqslant 2m$
- a strong decrease of matrix degree: $\deg(\bar{\mathbf{A}}) \leqslant 2\frac{D}{m}$
- preservation of the features targeted by our computations

examples:

- product $\mathbf{AB}$ easily deduced from product $\bar{\mathbf{A}}\bar{\mathbf{B}}$
- preservation of the determinant $\det(\mathbf{A}) = \det(\bar{\mathbf{A}})$
- inverse of $\bar{\mathbf{A}}$ contains inverse of $\mathbf{A}$ as submatrix
- ...

reduce **unbalanced** degrees to some **average** degree

basic illustration:

- let $\mathbf{A} \in \mathbb{K}[x]^{m \times m}$ of degree $< d$,
- let $\mathbf{u} \in \mathbb{K}[x]^{m \times 1}$ of degree $< md$,

then the matrix-vector product $\mathbf{Au}$ can be computed in

$\mathsf{MM}(m, d) + O(m^2 d)$ operations in $\mathbb{K}$

what would be the cost of the "naive" multiplication? $\rightsquigarrow O(m^2 \mathsf{M}(md))$

algorithm:

[Lecerf 2001 (in communication + software)]

first algorithms

partial linearization techniques

reduce **unbalanced** degrees to some **average** degree

basic illustration:

- let $\mathbf{A} \in \mathbb{K}[x]^{m \times m}$ of degree $< d$,
- let $\mathbf{u} \in \mathbb{K}[x]^{m \times 1}$ of degree $< md$,

then the matrix-vector product $\mathbf{Au}$ can be computed in

$$\mathsf{MM}(m, d) + O(m^2 d) \text{ operations in } \mathbb{K}$$

what would be the cost of the "naive" multiplication? $\rightsquigarrow O(m^2 \mathsf{M}(md))$

algorithm:

[Lecerf 2001 (in communication + software)]

$$\begin{bmatrix} \\ & \mathbf{A} & \\ \\ \end{bmatrix} \begin{bmatrix} \\ \mathbf{u} \\ \\ \end{bmatrix} = \begin{bmatrix} \\ & \mathbf{A} & \\ \\ \end{bmatrix} \begin{bmatrix} \\ & \bar{\mathbf{U}} & \\ \\ \end{bmatrix} \begin{bmatrix} 1 \\ x^d \\ x^{2d} \\ \vdots \end{bmatrix}$$

where the columns of $\bar{\mathbf{U}} \in \mathbb{K}[x]^{m \times m}$ form the x^d-adic expansion of $\mathbf{u}$

$\Rightarrow$ here $\deg(\bar{\mathbf{U}}) < d$

first algorithms

partial linearization techniques

reduce **unbalanced** degrees to some **average** degree

basic illustration:

- let $\mathbf{A} \in \mathbb{K}[x]^{m \times m}$ of degree $< d$,
- let $\mathbf{u} \in \mathbb{K}[x]^{m \times 1}$ of degree $< md$,

then the matrix-vector product $\mathbf{Au}$ can be computed in

$$\mathsf{MM}(m, d) + O(m^2 d) \text{ operations in } \mathbb{K}$$

what would be the cost of the "naive" multiplication? $\rightsquigarrow O(m^2 \mathsf{M}(md))$

algorithm:

[Lecerf 2001 (in communication + software)]

m	d	md	via `PolMatMul`	matrix-vector
10	20000	200000	0.203	0.368
20	5000	100000	0.225	0.683
40	2500	100000	0.528	2.481
80	1250	100000	1.227	9.592

outline

- **computer algebra**
  - efficient algorithms and software
  - for matrices over a field
  - for univariate polynomials
- **polynomial matrices**
  - basic definitions and properties
  - use in various situations
  - seen as matrices / seen as polynomials
- **first algorithms**
  - exploiting evaluation-interpolation
  - extending algorithms for polynomials
  - partial linearization techniques
- **exercises**

outline

computer algebra

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

polynomial matrices

- basic definitions and properties
- use in various situations
- seen as matrices / seen as polynomials

first algorithms

- exploiting evaluation-interpolation
- extending algorithms for polynomials
- partial linearization techniques

exercises

- evaluation-interpolation-based algorithms
- Krylov iterates via repeated squaring
- Krylov iterates in `MatMul` time

exercises

evaluation-interpolation-based algorithms

exercise: multiplication, determinant, inversion

1. adapting the evaluation-interpolation paradigm to matrices in $\mathbb{K}[x]^{m \times m}$,

- give an explicit **multiplication** algorithm
- give a **determinant** algorithm
- give an **inversion** algorithm

computing the inverse over the fractions $\mathbb{K}(x)$

2. for each of these algorithms,

- give a required lower bound on the **cardinality of $\mathbb{K}$**
- state and prove an upper bound on the **complexity**

hint: use known degree bounds on the output

evaluation-interpolation: multiplication

given $\mathbf{A}$ and $\mathbf{B}$ in $\mathbb{K}[x]^{m\times m}$ of degree $\leqslant d$,
we know that $\mathbf{C} = \mathbf{AB}$ has degree at most $2d$, so:

1. pick points: pairwise distinct $\alpha_1, \ldots, \alpha_{2d+1} \in \mathbb{K}$ — $\mathsf{Card}(\mathbb{K}) \geqslant 2d+1$
2. evaluate: $\mathbf{A}(\alpha_i)$ and $\mathbf{B}(\alpha_i)$, for $i = 1, \ldots, 2d+1$ — $O(m^2\mathsf{M}(d)\log(d))$
3. multiply: $\mathbf{A}(\alpha_i)\mathbf{B}(\alpha_i)$, for $i = 1, \ldots, 2d+1$ — $O(m^\omega d)$
4. interpolate: find $\mathbf{C}$ in $\mathbb{K}[x]^{m\times m}$ of degree $\leqslant 2d$ such that $\mathbf{C}(\alpha_i) = \mathbf{A}(\alpha_i)\mathbf{B}(\alpha_i)$, for $i = 1, \ldots, 2d+1$ — $O(m^2\mathsf{M}(d)\log(d))$
5. return $\mathbf{C}$

excellent algorithm:
. linear in d in the term $m^\omega d$ (recall Cantor-Kaltofen: $m^\omega d\log(d)$)
. exponent ω of matrix multiplication
. the $m^2\mathsf{M}(d)\log(d)$ term can be improved via points in geometric sequence
. downside: restriction on $\mathbb{K}$ (large degrees + small finite fields does happen)

evaluation-interpolation: determinant

given $\mathbf{A}$ in $\mathbb{K}[x]^{m\times m}$ of degree $\leqslant d$,
we know that $\Delta = \det(\mathbf{A})$ has degree at most md, so:

1. pick points: pairwise distinct $\alpha_1, \ldots, \alpha_{md+1} \in \mathbb{K}$ — $\mathrm{Card}(\mathbb{K}) \geqslant md+1$
2. evaluate: $\mathbf{A}(\alpha_i)$ for $i = 1, \ldots, md+1$ — $O(m^3 \mathsf{M}(d)\log(d))$
3. determinant: $\beta_i = \det(\mathbf{A}(\alpha_i))$, for $i = 1, \ldots, md+1$ — $O(m^{\omega+1}d)$
4. interpolate: find Δ in $\mathbb{K}[x]$ of degree $\leqslant md$ such that $\Delta(\alpha_i) = \beta_i$, for $i = 1, \ldots, md+1$ — $O(\mathsf{M}(md)\log(md))$
5. return Δ

- quasi-linear in degree d: fast for large d, small m
- exponent > 3 on matrix dimension m: slow for large m
- best known today: $\tilde{O}(m^\omega d)$

exercises

evaluation-interpolation: inversion

given $\mathbf{A}$ in $\mathbb{K}[x]^{m\times m}$ of degree $\leqslant d$,
we know that $\mathbf{C} = \mathbf{A}^{-1} = \frac{1}{\Delta}\mathbf{U}$ with
$\deg(\Delta) \leqslant md$ and $\deg(\mathbf{U}) \leqslant (m-1)d$, so:

0. set $n = (2m-1)d+1$ — $n = \Theta(md)$
1. pick points: pairwise distinct $\alpha_1, \ldots, \alpha_n \in \mathbb{K}$ — $\mathrm{Card}(\mathbb{K}) \geqslant (2m-1)d+1$
2. evaluate: $\mathbf{A}(\alpha_i)$, for $i = 1, \ldots, n$ — $O(m^3 M(d) \log(d))$
3. invert: $\mathbf{A}(\alpha_i)^{-1}$, for $i = 1, \ldots, n$ — $O(m^{\omega+1} d)$
4. interpolate: using Cauchy interpolation find $\mathbf{C}$ in $\mathbb{K}(X)^{m\times m}$ with all numerators of degree $\leqslant (m-1)d$ and all denominators of degree $\leqslant md$ such that $\mathbf{C}(\alpha_i) = \mathbf{A}(\alpha_i)^{-1}$, for $i = 1, \ldots, n$ — $O(m^2 M(md) \log(md))$
5. return $\mathbf{C}$

- quasi-linear in degree d: fast for large d, small m
- exponent > 3 on dimension m but recall size of $\mathbf{A}^{-1}$ is typically $\Theta(m^3 d)$
- best known today: $\tilde{O}(m^3 d)$, and even $\tilde{O}(m^\omega d)$ for factorized form
- note: one could compute $\det(\mathbf{A})$ to avoid Cauchy interpolation

exercises

problem (Krylov iterates):

input: matrix $A \in \mathbb{K}^{m\times m}$,

vector $v \in \mathbb{K}^{m\times 1}$

integer $d > 0$

output: $v, Av, \ldots, A^{d-1}v$

kernel black box:
given a matrix $\mathbf{F} \in \mathbb{K}[x]^{m\times(m+1)}$ of rank m and degree $\leqslant 1$, one can compute a nonzero element of degree $\leqslant m$ in the right kernel of $\mathbf{F}$ using $O(m^\omega)$ operations in $\mathbb{K}$

[refined analysis of Algo.1 in Zhou-Labahn-Storjohann 2012]

1. give an algorithm which costs $O(m^\omega \log(d) + m^{\omega-1}d)$ operations in $\mathbb{K}$, based on repeated squaring
2. prove that the generating series of $(A^k v)_{k\geqslant 0}$ rewrites as a fraction of polynomial matrices:

$$\sum_{k\geqslant 0} A^k v\, x^k = (I - xA)^{-1} v$$

3. using the kernel black box, give a complexity bound for finding $\lambda \in \mathbb{K}[x]$ and $\mathbf{u} \in \mathbb{K}[x]^{m\times 1}$, both of degree $\leqslant m$, such that

$$\sum_{k\geqslant 0} A^k v\, x^k = \mathbf{u}/\lambda$$

4. show that $(A^k v)_{0\leqslant k<d}$ can be computed in $O(m^\omega + mM(d))$

exercises

problem (Krylov iterates):

input: matrix $A \in \mathbb{K}^{m \times m}$,
vector $v \in \mathbb{K}^{m \times 1}$
integer $d > 0$

output: $v, Av, \ldots, A^{d-1}v$

kernel black box:
given a matrix $\mathbf{F} \in \mathbb{K}[x]^{m \times (m+1)}$ of rank m and degree $\leqslant 1$, one can compute a nonzero element of degree $\leqslant m$ in the right kernel of $\mathbf{F}$ using $O(m^\omega)$ operations in $\mathbb{K}$

[refined analysis of Algo.1 in Zhou-Labahn-Storjohann 2012]

1. give an algorithm which costs $O(m^\omega \log(d) + m^{\omega-1}d)$ operations in $\mathbb{K}$, based on repeated squaring

for simplicity, take d a power of 2

first compute $A^2, A^4, \ldots, A^{d/2}$, cost $O(m^\omega \log(d))$

from v, compute Av
from $[v \;\; Av]$, compute $A^2[v \;\; Av] = [A^2v \;\; A^3v]$
from $[v \;\; Av \;\; A^2v \;\; A^3v]$, compute $A^4[v \;\; Av \;\; A^2v \;\; A^3v] = [A^4v \;\; A^5v \;\; A^6v \;\; A^7v]$
etc...
from $[A^kv]_{0 \leqslant k < d/2}$, compute $A^{d/2}[A^kv]_{0 \leqslant k < d/2} = [A^kv]_{d/2 \leqslant k < d}$

exercises

problem (Krylov iterates):

input: matrix $A \in \mathbb{K}^{m \times m}$,
vector $v \in \mathbb{K}^{m \times 1}$
integer $d > 0$

output: $v, Av, \ldots, A^{d-1}v$

kernel black box:
given a matrix $\mathbf{F} \in \mathbb{K}[x]^{m \times (m+1)}$ of rank m and degree $\leqslant 1$, one can compute a nonzero element of degree $\leqslant m$ in the right kernel of $\mathbf{F}$ using $O(m^\omega)$ operations in $\mathbb{K}$

[refined analysis of Algo.1 in Zhou-Labahn-Storjohann 2012]

1. give an algorithm which costs $O(m^\omega \log(d) + m^{\omega-1}d)$ operations in $\mathbb{K}$, based on repeated squaring

the first $\min(\log(d), \log(m))$ products involve matrices of dimensions m or less, hence a total cost bounded by $O(m^\omega \log(d))$

the remaining products (if any) involve a lefthand operand of dimensions $m \times m$ and a righthand one of dimensions $m \times 2^k$, where k goes from about $\log_2(m)$ to for $\log_2(d)$
$\rightsquigarrow$ for a given k, the product costs $O(m^{\omega-1}2^k)$
$\rightsquigarrow$ summing this over all k, with $\sum_{k \leqslant \log_2(d)} 2^k \in O(d)$, gives $O(m^{\omega-1}d)$

exercises

problem (Krylov iterates):

input: matrix $A \in \mathbb{K}^{m \times m}$,

vector $v \in \mathbb{K}^{m \times 1}$

integer $d > 0$

output: $v, Av, \ldots, A^{d-1}v$

kernel black box:
given a matrix $\mathbf{F} \in \mathbb{K}[x]^{m \times (m+1)}$ of rank m and degree $\leqslant 1$, one can compute a nonzero element of degree $\leqslant m$ in the right kernel of $\mathbf{F}$ using $O(m^\omega)$ operations in $\mathbb{K}$

[refined analysis of Algo.1 in Zhou-Labahn-Storjohann 2012]

2. prove that the generating series of $(A^k v)_{k \geqslant 0}$ rewrites as a fraction of polynomial matrices:

$$\sum_{k \geqslant 0} A^k v x^k = (I - xA)^{-1} v$$

multiply the left-hand side by $I - xA$, this yields v

exercises

problem (Krylov iterates):

input: matrix $A \in \mathbb{K}^{m\times m}$,

vector $v \in \mathbb{K}^{m\times 1}$

integer $d > 0$

output: $v, Av, \ldots, A^{d-1}v$

kernel black box:
given a matrix $\mathbf{F} \in \mathbb{K}[x]^{m\times(m+1)}$ of rank m and degree $\leqslant 1$, one can compute a nonzero element of degree $\leqslant m$ in the right kernel of $\mathbf{F}$ using $O(m^\omega)$ operations in $\mathbb{K}$

[refined analysis of Algo.1 in Zhou-Labahn-Storjohann 2012]

3. using the kernel black box, give a complexity bound for finding $\lambda \in \mathbb{K}[x]$ and $\mathbf{u} \in \mathbb{K}[x]^{m\times 1}$, both of degree $\leqslant m$, such that

$$\sum_{k\geqslant 0} A^k v\, x^k = \mathbf{u}/\lambda$$

. consider $\mathbf{F} = [I - xA \quad -v]$; this matrix has degree $\leqslant 1$ and rank m (its leftmost $m\times m$ submatrix is nonsingular)

. so, in $O(m^\omega)$, we can compute a nonzero element of degree $\leqslant m$ in its right kernel

. this element can be written $\left[\begin{smallmatrix}\mathbf{u}\\ \lambda\end{smallmatrix}\right]$, and $\mathbf{F}\left[\begin{smallmatrix}\mathbf{u}\\ \lambda\end{smallmatrix}\right] = 0$ rewrites as $(I - xA)\mathbf{u} = v\lambda$

. observe that λ cannot be zero (otherwise, $\mathbf{u}$ would be a nonzero vector in the right kernel of $I - xA$, which is not possible)

. thus $(I - xA)^{-1}v = \frac{1}{\lambda}\mathbf{u}$

problem (Krylov iterates):

input: matrix $A \in \mathbb{K}^{m\times m}$,

vector $v \in \mathbb{K}^{m\times 1}$

integer $d > 0$

output: $v, Av, \ldots, A^{d-1}v$

kernel black box:
given a matrix $\mathbf{F} \in \mathbb{K}[x]^{m\times(m+1)}$ of rank m and degree $\leqslant 1$, one can compute a nonzero element of degree $\leqslant m$ in the right kernel of $\mathbf{F}$ using $O(m^\omega)$ operations in $\mathbb{K}$

[refined analysis of Algo.1 in Zhou-Labahn-Storjohann 2012]

4. show that $(A^k v)_{0\leqslant k<d}$ can be computed in $O(m^\omega + mM(d))$

. these d vectors are the first d terms of the series $\sum_{k\geqslant 0} A^k v\, x^k$
. we have seen that this series is equal to $\frac{1}{\lambda}\mathbf{u}$ (with $\mathbf{u}$ and λ found in $O(m^\omega)$)
$\rightsquigarrow$ it suffices to expand $\mathbf{u}/\lambda$ as a power series in precision d
. since $\mathbf{u}$ is a vector of m entries, this costs $O(mM(d))$

summary

computer algebra

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

polynomial matrices

- basic definitions and properties
- use in various situations
- seen as matrices / seen as polynomials

first algorithms

- exploiting evaluation-interpolation
- extending algorithms for polynomials
- partial linearization techniques

exercises

- evaluation-interpolation-based algorithms
- Krylov iterates via repeated squaring
- Krylov iterates in `MatMul` time