Exact linear algebra over the complex numbers

Fredrik Johansson
Inria Bordeaux

Applications of Computer Algebra (ACA)
July 23, 2021 (Online)
Session on Symbolic and Exact Linear Algebra over Rings and Fields

Representations

Inexact

- Floating-point arithmetic
- Ball arithmetic

Example: $\det(A) \in [0.23 \pm 0.01] \not\ni 0 \implies \operatorname{rank}(A) = n$
Example: $(\det(A) \in \mathbb{Z}) \in [0.99 \pm 0.02] \implies \det(A) = 1$

Exact

- Rational numbers
- Algebraic numbers
- Transcendental numbers

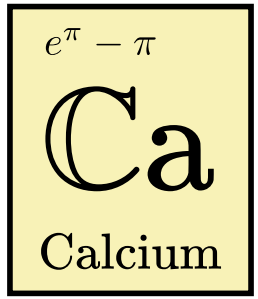
Exact fields of complex numbers

$$\mathbb{Q}(a_1, \ldots, a_n) \cong \text{Frac}(\mathbb{Q}[X_1, \ldots, X_n]/I)$$

Extension numbers a_k: $\sqrt{2}$, π, $\log(1 + \sqrt{2}\pi i)$, $\ldots$

- The trivial field $K = \mathbb{Q}$
- Transcendental fields $K = \mathbb{Q}(a_1, \ldots, a_n) \cong \mathbb{Q}(X_1, \ldots, X_n)$
- Algebraic number fields $K = \mathbb{Q}(a) \cong \mathbb{Q}[X]/\langle f(X) \rangle$
- Multivariate algebraic number fields
- Mixed fields. $K = \mathbb{Q}(\log(i), \pi, i) \cong \text{Frac}(\mathbb{Q}[X_1, X_2, X_3]/I)$, $I = \langle 2X_1 - X_2X_3, X_3^2 + 1 \rangle$

Linear algebra in Calcium[1]



`https://fredrikj.net/calcium/`

C library for exact real and complex numbers. Supports polynomials, matrices.

Includes a Python interface. Julia interface (in Nemo.jl) in progress.

ISSAC paper: `https://arxiv.org/abs/2011.01728`

[1] Not to be confused with the chemical element with atomic number 20. $e^\pi - \pi = 19.999099979\ldots$

Need **rigorous numerics** and **rigorous algebra**

$$a = 2\log(\sqrt{2}+\sqrt{3}) - \log(5+2\sqrt{6}) \qquad (a = 0)$$

$$A = \begin{pmatrix} 0 & a \\ 0 & 0 \end{pmatrix} \quad B = \begin{pmatrix} 0 & a + e^{-1000} \\ 0 & 0 \end{pmatrix}$$

Maple 2020, SageMath 9.2 `SymbolicRing`: $\operatorname{rank}(A) = 1$

Mathematica 12.2: $\operatorname{rank}(B) = 0$

Calcium:

```
>>> a = 2*log(sqrt(2)+sqrt(3)) - log(5+2*sqrt(6))
>>> A = ca_mat([[0,a],[0,0]]); A.rank()
0
>>> B = ca_mat([[0,a+exp(-1000)],[0,0]]); B.rank()
1
```

Fundamental operations

- Multiplication
- Solving $Ax = b$, LU factorization
- Inverse/adjugate
- Determinant, rank
- Characteristic polynomial
- Eigenvalues, Jordan normal form, exp/log, ...

Typical reductions:

- Solving, inverse, rank, determinant $\to$ LU factorization
- LU factorization $\to$ matrix multiplication
- Anything $\to$ characteristic polynomial
- Characteristic polynomial $\to$ matrix multiplication
- Anything $\to$ matrix multiplication

Division-avoiding algorithms

Why avoid divisions?

- May be unable to test $z = 0$
- May be expensive to test $z = 0$
- Denominator explosion / expensive to simplify fractions

Various algorithms:

- Fewer or "exact" divisions only (e.g. fraction-free Gauss)
- Integer divisions only (e.g. Faddeev-Leverrier)
- Completely division-free (e.g. Berkowitz)

Example: a determinant

Compute

$$\det \begin{pmatrix} \sqrt{1} & \sqrt{2} & \sqrt{3} & \sqrt{4} & \sqrt{5} \\ \sqrt{2} & \sqrt{3} & \sqrt{4} & \sqrt{5} & \sqrt{6} \\ \sqrt{3} & \sqrt{4} & \sqrt{5} & \sqrt{6} & \sqrt{7} \\ \sqrt{4} & \sqrt{5} & \sqrt{6} & \sqrt{7} & \sqrt{8} \\ \sqrt{5} & \sqrt{6} & \sqrt{7} & \sqrt{8} & \sqrt{9} \end{pmatrix}$$

Calcium will perform arithmetic in

$\mathbb{Q}(\sqrt{7}, \sqrt{6}, \sqrt{5}, \sqrt{3}, \sqrt{2}) \stackrel{?}{\cong}$
$\mathrm{Frac}(\mathbb{Q}[a, b, c, d, e]/\langle a^2 - 7, b^2 - 6, c^2 - 5, d^2 - 3, e^2 - 2, b - de \rangle)$

Example: a determinant

Gaussian elimination (LU factorization):

```
(156829688*a*c*d*e-221693656*a*c*d+271638392*a*c*e-383986048*a*c
    +274164856*a*d*e-387945384*a*d+474865368*a*e-671936784*a+361353464*
    c*d*e-510531104*c*d+625886152*c*e-884270248*c+959654264*d*e
    -1358274640*d+1662163432*e-2352590040) / (18200*a*c*d*e-25732*a*c*d
    +31512*a*c*e-44565*a*c+324056*a*d*e-458284*a*d+561288*a*e-793807*a
    +847420*c*d*e-1198107*c*d+1467772*c*e-2075132*c+1068396*d*e
    -1511729*d+1850596*e-2618400)
```

Fraction-free Gaussian elimination (FFLU):

```
(-28*a*c*d*e+48*a*c*d+20*a*c*e-116*a*c+460*a*d*e-520*a*d+332*a*e-532*a
    +348*c*d*e-516*c*d-332*c*e+120*c+548*d*e-388*d+1660*e-2144) / (c*d
    -2*c+4*d*e-3*d-4)
```

Cofactor expansion, Faddeev-Leverrier or Berkowitz algorithm:

```
-4*a*c*d-20*a*c*e-24*a*c-4*a*d*e+8*a*d+136*a-28*c*d*e-116*c*d-88*c*e+64*
    c+112*d*e+164*d-60*e+244
```

Faddeev-Leverrier algorithm

The coefficients of charpoly(A) satisfy:

$$c_{n-k} = -\frac{1}{k} \sum_{j=1}^{k} c_{n-k+j} \operatorname{Tr}(A^j)$$

Output: (charpoly(A), det(A), adj(A))

Requires computing A^2, A^3, ..., A^n
Complexity: $O(n^4)$ classically, or $O(nM(n))$

Not widely used. Why?

- Berkowitz algorithm is $O(n^4)$ with smaller constant factor
- Numerically unstable

Improved F-L algorithm (Preparata-Sarwate algorithm)

Observation: $\mathrm{Tr}(AB)$ can be computed using $O(n^2)$ operations

Compute $A^1, A^2, \ldots, A^m$ and $A^m, A^{2m}, A^{3m}, \ldots$ with $m \approx \sqrt{n}$

Complexity: $O(n^{3.5})$ classically, or $O(n^{0.5} M(n) + n^3)$

Not widely known. Some history:

- Preparata and Sarwate (1978)
- Berkowitz (1984) citing private communication with Winograd
- Galil and Pan (1989) reduce the $O(n^3)$ term
- FJ (2020), `https://arxiv.org/abs/2011.12573`

Some implementation results

Practical fast matrix multiplication

Over $\mathbb{Z}$, $\mathbb{Q}$ (in FLINT, similarly in LinBox, Magma, ...):

- Strassen's algorithm
- Multimodular reduction $\to \mathbb{Z}/p\mathbb{Z} \to$ reconstruction

Number fields (used in Calcium):

- Bit packing: $\mathbb{Q}(\alpha) \to \mathbb{Z}[X] \to \mathbb{Z}$

Floating-point / ball arithmetic (used in Arb):[2]

- Optimized dot products
- Blocks + scaling: $\mathbb{Z}[\frac{1}{2}] \to \mathbb{Z}$

[2] FJ. *Faster arbitrary-precision dot product and matrix multiplication*, ARITH26 (2019) - `https://arxiv.org/abs/1901.04289`

Some implementation results

Empirical comparison of algorithms for:

- Determinant
- Adjugate/inverse
- Characteristic polynomial

Implemented in:

- FLINT (integers, rationals)
- Calcium
- Arb (ball arithmetic)

Linear algebra: integers/rationals

$n \times n$ matrix over $\mathbb{Z}$ with random elements in $-10, \ldots, 10$

n	Modular			FFLU		B	F-L	P-S
	D	DA	CD	D	DA	CD	CDA	CDA
				(in milliseconds)				
10	0.021	0.12	0.016	0.0060	0.015	0.035	0.015	0.030
20	0.078	0.96	0.11	0.036	0.43	1.0	0.86	0.61
				(in seconds)				
50	0.0012	0.016	0.0039	0.0023	0.011	0.048	0.052	0.017
100	0.0068	0.18	0.055	0.039	0.14	0.84	1.1	0.22
200	0.044	1.8	0.89	0.64	2.3	16	27	4.1
300	0.15	9.0	4.6	3.4	13	94	174	20
400	0.38	22	15	12	45	321	696	66
500	0.77	52	37	32	127	900	2057	150

C - characteristic polynomial
D - determinant
A - adjugate/inverse

FFLU - fraction-free Gauss
B - Berkowitz
F-L - Faddeev-Leverrier
P-S - Preparata-Sarwate

Linear algebra: number fields

$n \times n$ over $\mathbb{Q}(\zeta_{20})$, entries $\sum_k \frac{p}{q}\zeta_{20}^k$, random $|p| \leq 10$, $1 \leq q \leq 10$

n	Mod* CD	Hess CD	Dani CD	LU D	FFLU D	LU DA	FFLU DA	B CD	F-L CDA	P-S CDA
					(in seconds)					
10	0.038	0.31	0.16	0.024	0.0059	0.21	0.11	0.010	0.0073	0.010
20	0.12	19	6.7	0.22	0.067	2.6	1.4	0.28	0.15	0.16
40	1.1		353	2.8	0.9	37	22	7.5	3.7	2.6
60	3.4			15	4.7	182	119	54	19	12
80	7.5			53	15	581	409	208	67	34
100	15			144	41			608	670	130
120	24			322	83			1439	3013	420

Hess - Hessenberg algorithm ($O(n^3)$)
Dani - Danilevsky algorithm ($O(n^3)$)

* Modular algorithm in Sage (everything else implemented in Calcium)

Linear algebra: number fields

$n \times n$ DFT matrix over $\mathbb{Q}(\zeta_n)$, entries $A_{i,j} = \zeta_n^{(i-1)(j-1)}$

n	Mod* CD	Hess CD	Dani CD	LU D	FFLU D	LU DA	FFLU DA	B CD	F-L CDA	P-S CDA
					(in milliseconds)					
10	10	1.8	1.6	0.17	0.22	0.76	1.4	0.61	0.75	0.59
20	3.9	1.9	2.4	1.7	4.6	7.1	38	20	20	0.70
					(in seconds)					
50	1.3	0.17	0.13	0.065	0.80	0.28	6.0	8.2	2.0	0.49
100	22	5.4	22	0.89	43	5.3	335	803	223	29
150	78	22	7.9	4.4	214	19	1423	7259	933	138
200	333	1928	140	31	1655	192				1687

Hess - Hessenberg algorithm ($O(n^3)$)
Dani - Danilevsky algorithm ($O(n^3)$)

* Modular algorithm in Sage (everything else implemented in Calcium)

Linear algebra: multivariate fields

$n \times n$ matrix, entries $A_{i,j} = \sqrt{i + j - 1}$, $1 \le i, j \le n$

n	Hess CD	Dani CD	LU D	FFLU D	LU DA	FFLU DA	B CD	F-L CDA	P-S CDA
				(in milliseconds)					
3	0.18	0.25	0.076	0.050	0.48	0.60	0.055	0.20	0.15
4	2.3	2.2	0.40	0.24	3.7	3.3	0.37	1.3	1.4
5	242	7.8	31	0.63	129	11	1.1	4.1	4.0
				(in seconds)					
6	15	0.18	0.090	0.019	0.70	0.76	0.0033	0.014	0.016
7		0.99	0.25	0.064	3.3	2.1	0.0089	0.037	0.044
8		6.0	0.69	0.22	22	9.2	0.024	0.12	0.17
9				0.47		66	0.064	0.30	0.60
10				1.1			0.18	0.77	2.1
11							0.40	1.4	5.1
12							0.85	3.4	13
15							5.5	21	
20							188		

General observations

- Different algorithms good for different matrices
- Strong expression/fraction simplification usually more desirable than raw speed
- LU or FFLU good with simple ideal (e.g. $\mathbb{Q}(\sqrt{2})$, $\mathbb{Q}(\pi)$), Berkowitz otherwise (e.g. $\mathbb{Q}(\sqrt{2}, \sqrt{3})$)
- In the multivariate setting, Preparata-Sarwate seems to perform worse than the classical Faddeev-Leverrier algorithm. Reason: much faster to multiply by A (simple entries) than A^m (complicated entries).

Linear algebra: ball arithmetic

Uniformly random $n \times n$ real matrix

n	HessG CD	HessH CD	Dani CD	Eig CD	LU D	LU2 D	B CD	F-L CDA	P-S CDA
				(in milliseconds)					
10	0.21	0.38	0.22	17	0.068	0.23	0.80	1.0	0.78
20	2.1	3.2	2.0	180	0.52	1.5	3.9	17	9.2
				(in seconds)					
50	0.045	0.057	0.048	4.6	0.0078	0.019	0.22	0.61	0.21
100	0.64	0.69	0.61	56	0.062	0.15	6.3	9.7	2.4
150	3.5	3.5	3.0	245	0.23	0.44	52	52	11
200	12	11	10		0.59	1.0	224	176	34
250	31	29	25		1.4	1.9	687	460	73
300	66	59	53		2.5	3.2	1804	1075	160
350	135	115	110		4.4	5.0	4033	2107	306
p*	$10n$	$6n$	$10n$	0	n	0	$6n$	$6n$	$6n$

* Precision $333 + p$ chosen to give roughly 100-digit output accuracy

HessG/HessH - Hessenberg using Gauss/Householder
LU2 - approximate LU followed by a posteriori certification
Eig - approximate eigenvals $\rightarrow$ certification $\rightarrow$ charpoly reconstruction

Observations - ball arithmetic

If precision is precious, use certification methods.

Otherwise, try LU (determinant/inverse) or Hessenberg (charpoly).
If it fails, fall back on Berkowitz (small matrices) or
Preparata-Sarwate (large matrices).

Topics for further study

- Fast multivariate multiplication
- Algorithms for sparse matrices
- Modular methods (need to guarantee correct zero tests)
- Multivariate expression simplification in the context of linear algebra algorithms