MAGMA:
Advanced Matrix Algorithms

Allan Steel
University of Sydney
(Magma Project)

Overview

- Fast multiplication.
- Fast echelonization, determinant, rank, inverse over general fields.
- Fast nullspace, determinant, etc., over $\mathbf{Z}/\mathbf{Q}$.
- Fast Hermite Normal Form.

All timings and crossovers in this talk are for a 400Mhz Ultrasparc.

Convention:

When echelonizing X, a *transformation matrix* T is an invertible matrix such that

$$TX = E,$$

where E is the reduced row echelon (RRE) form of X.

Matrix Multiplication

Multiplication of a pair of 2 by 2 matrices:

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} = \begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix}$$

Classical Method: 8 multiplications, 4 additions.

$$c_{11} = a_{11}b_{11} + a_{12}b_{21},$$

$$c_{12} = a_{11}b_{12} + a_{12}b_{22},$$

$$c_{21} = a_{21}b_{11} + a_{22}b_{21},$$

$$c_{22} = a_{21}b_{12} + a_{22}b_{22}.$$

General Complexity: $O(n^3)$.

Strassen's Method (1969): 7 multiplications, 18 additions or subtractions (18 improved to 15 by Winograd).

Let:

$$x_1 = (a_{11} + a_{22}) \cdot (b_{11} + b_{22}),$$

$$x_2 = (a_{21} + a_{22}) \cdot b_{11},$$

$$x_3 = a_{11} \cdot (b_{12} - b_{22}),$$

$$x_4 = a_{22} \cdot (b_{21} - b_{11}),$$

$$x_5 = (a_{11} + a_{12}) \cdot b_{22},$$

$$x_6 = (a_{21} - a_{11}) \cdot (b_{11} + b_{12}),$$

$$x_7 = (a_{12} - a_{22}) \cdot (b_{21} + b_{22}).$$

Then:

$$c_{11} = x_1 + x_4 - x_5 + x_7,$$

$$c_{12} = x_3 + x_5,$$

$$c_{21} = x_2 + x_4,$$

$$c_{22} = x_1 + x_3 - x_2 + x_6.$$

- Strassen's method leads to a **recursive algorithm** for matrix multiplication – commutativity is NOT used!
- Implementation is very complicated for non-square matrices, and for dimensions which are not powers of 2.
- **Much** more difficult to implement than Karatsuba (the simplest asymptotically-fast methods for their respective problems).
- Obviously fast when multiplication in the base ring is expensive, but difficult to make fast otherwise.

Complexity: $O(n^{\log_2 7}) \approx O(n^{2.807})$.

Timings

Base ring K is **GF**(p), where $p = 11863279$ (for C double representation: $64p^2 \leq 2^{53}$)).

Dim	Classical	Strassen	Speedup
64	0.0065	0.0045	1.44
128	0.051	0.037	1.38
256	0.41	0.28	1.47
512	3.32	2.12	1.57
768	14.12	7.71	1.83
1024	55.43	15.15	3.66 [!?]
2048	436.90	106.95	4.09 [!?]

Discrepancy for 1024 and 2048 due to processor caching, etc. – Strassen accesses memory in a more efficient way, while the standard classical implementation is poor.

Thus spake the Master:

Donald Knuth, *AOCP*, Vol. 2, **3rd Ed., 1997** [NEW EDITION!], my emphasis:

> These **theoretical** results [Strassen's method] are quite striking, but from a practical standpoint they are of **little use** because n must be very large. . . [p. 501].
>
> Richard Brent (1970) estimated that Strassen's scheme would not begin to excel over Winograd's [cubic complexity] scheme until $n \approx$ **250**; and such **enormous** matrices **rarely** occur in practice unless they are very sparse, when other techniques apply [p. 501].
>
> **Of course** such asymptotically "fast" multiplication is **strictly of theoretical interest** [p. 718; added in 1997 edition!!!].

In response:

Erich Kaltofen, *Challenges of Symbolic Algebra* (EECAD 1998):

> Open Problem 7: Convince Donald Knuth that these asymptotically fast methods are of practical value. If he pays you $2.56 for this technical error, you have solved this problem.

Allan Steel (2000):

> For quite practical sizes, Strassen's method is **streets ahead** of the classical method. *[German etymological pun]*

Modular Matrix Multiplication

Multiply a pair of n by n matrices, with all entries being integers of up to k bits each.

$M(n)$: complexity of the matrix multiplication algorithm (arithmetic operations).

Assume k is small enough that only classical integer multiplication is applicable.

- **Classical method**: $M(n)O(k^2)$ bit operations.

- **Modular method**: $M(n)O(k)+O(n^2)O(k^2)$ bit operations.

  (Reduce the input modulo several primes, multiply each such pair modularly and use Chinese remaindering for the result.)

If $n >> k$, modular method is practically linear in k.

To multiply matrices over **GF**(p), large p: multiply over **Z** by above, then mod by p at end.

Recursive Echelonization Algorithm

Reduces echelonization to matrix multiplication so that the complexity is that of multiplication.

V. Strassen sketched such an algorithm for computing the inverse of a square matrix, assuming some conditions:

Gaussian Elimination is not Optimal (Numer. Math., 1969).

This paper also had the original fast multiplication formulae.

Idea: Recursive Inverse Algorithm

X is the input matrix; Y is the transformation matrix and will be the inverse at the end.

Assumes that intermediate matrices to be inverted are invertible.

$$X = \begin{pmatrix} A & B \\ C & D \end{pmatrix}, \quad Y = \begin{pmatrix} I & 0 \\ 0 & I \end{pmatrix}.$$

Compute A^{-1} recursively. Multiply the top half by A^{-1} (on the left) [1 mult].

$$X = \begin{pmatrix} I & A^{-1}B \\ C & D \end{pmatrix}, \quad Y = \begin{pmatrix} A^{-1} & 0 \\ 0 & I \end{pmatrix}.$$

Write $E = A^{-1}B$. Subtract C times the top half from the bottom half [2 mults].

$$X = \begin{pmatrix} I & E \\ 0 & D - CE \end{pmatrix}, \quad Y = \begin{pmatrix} A^{-1} & 0 \\ -CA^{-1} & I \end{pmatrix}.$$

Write $F = D - CE$. Compute the inverse of F recursively and multiply the bottom half by F^{-1} (on the left) [1 mult].

$$X = \begin{pmatrix} I & E \\ 0 & I \end{pmatrix}, \quad Y = \begin{pmatrix} A^{-1} & 0 \\ -F^{-1}CA^{-1} & F^{-1} \end{pmatrix}.$$

Write $G = -F^{-1}CA^{-1}$. Finally, subtract E times the bottom half from the top half [2 mults].

$$X = \begin{pmatrix} I & 0 \\ 0 & I \end{pmatrix}, \quad Y = \begin{pmatrix} A^{-1} - EG & -EF^{-1} \\ G & F^{-1} \end{pmatrix}.$$

Uses 6 half-size multiplications and 2 half-size inversions.

Even with classical multiplication, it is not slower than classical echelonization!

New general algorithm designed and implemented within Magma (Steel [2000]):

- Works on any shape of matrix with any rank.
- Benefits from **ANY** fast matrix multiplication algorithm (not just Strassen multiplication).
- Yields asymptotically-fast algorithms for reduced row echelon form, nullspace, determinant, inverse, etc.

KEY POINT: The base ring need not be a field with natural homomorphism into **GF**(p) for small p since inversions are not needed in a fast multiplication algorithm.

Works by similar idea, but with several complications:

- Zero rows, zero rows in submatrices, missing pivot columns.
- The rank of the first submatrix may not equal its number of rows.
- Pivot columns in second recursive echelon form may be to the left of those of first the first recursive echelon form.
- Determinant difficult to compute in nasty cases.

Complexity: $O\tilde{}(M(n))$, where $M(n)$ is complexity of matrix multiplication – we can use Strassen complexity in practice!

Timings

Large prime finite fields. Uses both modular multiplication **AND** Strassen's method for each small prime.

Over **GF**(p_{20}) without transformation matrix:

Dim	Classical	Recursive	Speedup
32	0.047	0.036	1.3
64	0.35	0.16	2.2
128	2.61	0.76	3.4
256	21.08	3.64	5.8
512	170.82	18.80	9.1
1024	1399.76	101.89	13.7

Over **GF**(p_{20}) with transformation matrix:

Dim	Classical	Recursive	Speedup
32	0.11	0.068	1.6
64	0.87	0.31	2.8
128	6.78	1.41	4.8
256	54.87	7.09	7.7
512	450.29	37.56	12.0
1024	3634.48	207.79	17.5

Small $K = \mathbf{GF}(p)$ where $p = 11863279$ [$64p^2 \leq 2^{53}$)]. Uses Strassen's method.

Over K without transformation matrix:

Dim	Classical	Recursive	Speedup
32	0.0011	0.0018	0.6
64	0.012	0.0080	1.5
128	0.097	0.038	2.6
256	0.79	0.23	3.4
512	6.46	1.44	4.5
1024	65.33	9.43	6.9
2048	578.45	63.67	9.1

Over K with transformation matrix:

Dim	Classical	Recursive	Speedup
32	0.0041	0.0030	1.4
64	0.035	0.013	2.7
128	0.28	0.067	4.2
256	2.29	0.43	5.3
512	20.44	2.89	7.1
1024	203.18	20.10	10.1
2048	1739.73	140.07	12.4

All subsequent algorithms in this talk use this!!!

Modular Nullspace over Z or Q

Several Primes/CRT method

- Compute the RRE form of the nullspace modulo each prime.
- Form the combined echelon form of the nullspace over **Q** using the Chinese remainder theorem (bad primes easily found).
- Apply rational reconstruction to each combined vector; if successful, check that it is a true integral null vector.
- Algorithm terminates when the number of correctly constructed vectors equals the modular nullity (which is an upper bound of the correct nullity).
- Saturate nullspace lattice over **Q** if result to be over **Z** [$(L \otimes \mathbf{Q}) \cap \mathbf{Z}^n$].

p-adic method

- Fix a prime p and compute the RRE form with transformation matrix of the input matrix A (which is over **Z**) modulo p **ONCE**. Uses fast echelonization algorithm, with Strassen complexity, blocked doubles.
- Using these matrices mod p, successively solve systems of the form $v_i{\cdot}A = w_i \pmod{p}$ to compute each null vector modulo p^i.
- Apply rational reconstruction to each null vector modulo p^i and test for correctness.
- Saturate nullspace lattice over **Q** if result to be over **Z**.

- The case that p is bad (p divides an elementary divisor of the input matrix) is easily detected (in the p-adic error-term calculation).
- One can do **each** null vector **independently**, and multiply by the LCM of previous common denominators to reduce the number of p-adic steps for later null vectors.
- Only **one modular echelonization** (with transformation matrix) need be performed, so this algorithm is much faster than the several-primes/CRT method if the nullity is small.
- Running time is proportional to the size of the entries of the **resulting nullspace** (not the size of the input entries).

Timings

Nullspaces over the integers of integral matrices with random $\{-1, 0, 1\}$ entries:

Dim	Null.	Hermite	CRT	p-adic
200	1	21.5	5.9	0.7
200	10	26.0	13.1	2.6
300	1	153.8	30.0	2.0
300	10	175.7	53.6	8.8
400	1	621.0	92.2	5.1
400	10	691.1	161.4	25.4
500	1	1904.5	246.6	11.8
500	10	2016.9	371.5	56.4
1000	1	?	4841.9	97.0

Heuristic Complexity for n by n random integral matrices with bounded bit length:

CRT algorithm: $O\tilde{~}(n{\cdot}M(n)) \approx O\tilde{~}(n^4)$.
p-adic algorithm: $O\tilde{~}(n^3)$.

Determinant of integral matrix X

Basic modular (CRT) algorithm:

Compute determinant modulo enough primes to cover Hadamard bound Had(X) and use Chinese Remainder Theorem.

Uses fast echelonization, Strassen complexity.

p-adic algorithm:

J. Abbot, M. Bronstein, T. Mulders, "Fast Deterministic Computation of Determinants of Dense Matrices" (ISSAC 1999).

Heuristic Complexity for n by n random integral matrices with bounded bit length:

CRT algorithm: $O\tilde{}(n{\cdot}M(n)) \approx O\tilde{}(n^4)$.
p-adic algorithm: $O\tilde{}(n^3)$.

p-adic algorithm:

- Choose random integral vector w (entries in range $\{-100\ldots100\}$, say).
- Solve system $v \cdot X = w$ over $\mathbf{Q}$ by p-adic nullspace algorithm. Thus $v = w \cdot X^{-1}$ over $\mathbf{Q}$.
- The common denominator d of v is a divisor of the common denominator C of X^{-1}. But C is the largest elementary divisor of X.
- Thus d divides the true determinant D; use modular algorithm to compute D/d with bound Had$(X)/d$.

Much faster if d is reasonably close to Had(X) (for random matrices C is close to true determinant).

Hermite Normal Form (HNF)

For a square non-singular matrix X, if H is the HNF of X then:

- H is row-equivalent to X,
- H is upper triangular, and all entries of H are non-negative,
- For each diagonal entry of H, the entries above it are less than it.

E.g.:

$$X = \begin{pmatrix} -5 & 2 \\ 6 & 7 \end{pmatrix}, \quad H = \begin{pmatrix} 1 & 9 \\ 0 & 47 \end{pmatrix}.$$

$$X = \begin{pmatrix} 2 & 3 & 2 \\ -6 & -6 & -11 \\ -4 & 2 & 11 \end{pmatrix}, \quad H = \begin{pmatrix} 2 & 0 & 7 \\ 0 & 3 & 35 \\ 0 & 0 & 40 \end{pmatrix}.$$

"Classical" Kannan-Bachem HNF algorithm

X is $m \times n$ matrix with entries in $\mathbf{Z}$, bound bit-length for entries.

At step k ($k = 1, \ldots, m$), ensure that the first k rows of X are the full HNF of the space generated by the first k rows.

This is done by simply 'inserting' row k into the space generated by the preceding rows (using elementary row operations).

Complexity $[m = n]$: $O\tilde{}(n^5)$.

New Modular HNF Algorithm

Based on heuristic idea of:

> Daniele Micciancio, Bogdan Warinschi: *A Linear Space Algorithm for Computing the Hermite Normal Form* [Proc. ISSAC 2001].

Let X be an n by n matrix with entries in $\mathbf{Z}$. Assume X has rank n. We wish to compute the Hermite Normal Form of X.

Write:

$$X = \left(\begin{array}{c|c} A & c \\ \hline v & v_c \\ \hline w & w_c \end{array}\right).$$

$A \in \mathbf{Z}^{(n-2)\times(n-1)}$,
$c \in \mathbf{Z}^{(n-2)\times 1}$,
$v, w \in \mathbf{Z}^{1\times(n-1)}$, $v_c, w_c \in \mathbf{Z}$.

Let

$$d_1 = \det\begin{pmatrix} A \\ \hline v \end{pmatrix}, \quad d_2 = \det\begin{pmatrix} A \\ \hline w \end{pmatrix},$$

$$g = \mathrm{GCD}(d_1, d_2), \quad g = a_1 d_1 + a_2 d_2.$$

Then

$$\det\begin{pmatrix} A \\ \hline a_1 v + a_2 w \end{pmatrix} \quad = \quad g.$$

So we can multiply the bottom two rows of X by:

$$T = \begin{pmatrix} a_1 & a_2 \\ \frac{-d_2}{g} & \frac{d_1}{g} \end{pmatrix}$$

which has determinant 1.

We obtain:

$$X_1 = \left(\begin{array}{c|c} B & d \\ \hline u & u_c \end{array}\right),$$

where the last row of B is $a_1 v + a_2 w$.

The determinant of B is g.

The Hermite form H of B (over $\mathbf{Z}$) equals the Hermite form of B mod $(2g)$. This can be computed very quickly, assuming g is small.

Now suppose U is the unique unimodular matrix (determinant 1 or -1), such that $H = UB$.

We want to replace B by H but also update the right column d of X_1 at the same time.

We want the column vector e such that:

$$U \cdot \left(B \,|\, d \right) \quad = \quad \left(H \,|\, e \right).$$

Obviously, $U = HB^{-1}$ and $e = Ud$.

If the transformation matrix is wanted for the full algorithm, then we compute $U = HB^{-1}$ via a modular algorithm and set $e = Ud$.

Otherwise, there is a much faster way to find e. We have:

$$e = Ud = HB^{-1}d, \quad \text{so} \quad BH^{-1}e = d.$$

Now H has entries up to at most g and has determinant g, so H^{-1} (over $\mathbf{Q}$) has very small entries so can be computed very quickly.

So we easily compute BH^{-1} (which is over **Z** since it equals U^{-1}) and simply solve the equation:

$$(BH^{-1}) \cdot e = d$$

for the vector e (using the p-adic nullspace algorithm), since the vector d is known.

Thus we obtain:

$$X_2 = \left(\begin{array}{c|c} H & e \\ \hline u & u_c \end{array} \right).$$

Finally, we do simple row operations to reduce the vector $(u|u_c)$ by the rows above it etc. to find the complete Hermite Normal Form of X.

Time dominated by:

- The computation of the 2 determinants d_1 and d_2 (BUT can be combined to take practically the time of 1 determinant).
- The modular Hermite form H of B and the inverse of H (very trivial if g small).
- The nullspace computation to solve for e in $(BH^{-1}) \cdot e = d$.

Thus: roughly one p-adic determinant plus one p-adic nullspace if g small.

Heuristic Complexity for n by n random integral matrices with bounded bit length: $O^{\sim}(n^3)$ – timings on last slide obey this!

General Algorithm

- The **true rank** r of X and a submatrix having rank r can be found during the initial p-adic determinant computation in the p-adic nullspace stage.
- If X has **more columns** than the rank:

  Then there are simply more columns for e and d in the above equation $(BH^{-1})e = d$ to be solved.
- If X has **more rows** than the rank:

  Then we use more vectors than the 2 vectors v and w to compute the GCD g (and we use LLL to get a good small combination of vectors). The reductions of the extra rows at the end are very fast.

Timings

Random integral matrices in range $\{-9 \ldots 9\}$ (except range $\{-1 \ldots 1\}$ for dimension 1000):

Dim	KB Hermite	New Herm	CRT Det	p-adic Det	Det Digs
100	2.6	0.5	0.6	0.3	153
150	17.0	1.5	2.3	0.8	241
200	65.1	3.4	7.0	1.8	334
250	189.2	6.3	17.0	3.7	430
300	452.6	11.1	28.9	5.8	528
350	953.9	17.0	55.5	9.5	629
400	1786.6	27.0	86.4	12.6	730
450	3111.4	36.7	147.6	19.6	830
500	5144.2	53.9	230.7	29.1	936
1000	55762.2	246.0	1782.2	171.3	1193

The new Hermite algorithm is much faster than the standard many-primes (CRT) determinant algorithm, which has been considered the easy "preprocessing" step to other modular Hermite algorithms!!!

[NB also: even CRT Det here uses the fast Strassen multiplication and echelonization.]