Linear Algebra Modulo Tiny Primes

David Saunders, Bryan Youse - U Delaware
Jean-Guillaume Dumas - U Grenoble

... using and extending the LinBox library.

www.linalg.org

Integer Rank and Rank Modulo Tiny Primes

David Saunders, Bryan Youse - U Delaware
Jean-Guillaume Dumas - U Grenoble

... using and extending the LinBox library.

www.linalg.org

Outline

- large matrix, small rank conjecture for Dickson SRG family.
- large matrix, large rank
- packing schemes
- small rank algorithm, resources needed
- large rank resources needed
- integration of packing into LinBox

Algebraic graph theory

Ranks and Smith forms of incidence and adjacency matrices play an important role in classification of various graphs and graph families.

Qing Xiang, Peter Sin, David Chandler, ... strongly regular graphs: Matrix order $n = 3^{16} \sim 43 \times 10^6$. 3-Rank to be computed $r < 2^{17} \sim 10^5$.

Previous case done: $n = 3^{14}, r = 32064$, 4 days single process.

Andries Brouwer, ... distance regular graphs: Matrix order $n \sim 10^6$. p-Rank to be computed $r \sim n$, smallish p.

Previous case done: $n \sim 10^5$, 1 file per row, amusing probs.

Dickson's Hadamard difference set

G = additive group of $\mathrm{GF}(p^e) \times G\mathrm{F}(p^e)$
Difference set D $= \{(a^2 + gb^{2\sigma}, 2ab) | (0,0) \neq (a,b) \in \mathrm{G}\}$
where g is a generator, σ an automorphism of $\mathrm{GF}(p^e)$.
(D is the set of non-zero squares of a semi-field multiplication on G).
Adjacency matrix:

$$a_{i,j} = \begin{cases} p-1, & \text{if } i = j, \\ 1, & \text{if } e_i - e_j \in D, \\ 0, & \text{otherwise.} \end{cases}$$

Dickson SRG					
e	$n = 3^{2e}$	r near 2^{2e}	2007t	2009r	2009t
1	9	4	-	-	-
2	81	20	0.021s	0.0003	0.0012s
3	729	85	0.35s	0.003s	0.022s
4	6,561	376	33.3s	0.046s	0.95s
5	59,049	1654	0.5h	1.4s	0.017h
6	531,441	7283	46.7h	80s	1.2h
7	4,782,969	32064	-	1.2h	96.4h
8	43,046,721	?	-	-	-

Table 1: The Dickson SRG example computed with summation and certificate. The time units are 's' for seconds, and 'h' for hours.

Hankel system

$$\begin{pmatrix} 4 & 20 & 85 & 376 \\ 20 & 85 & 376 & 1654 \\ 85 & 376 & 1654 & 7283 \\ & & & \end{pmatrix} \begin{pmatrix} 1 \\ -2 \\ -4 \\ 1 \end{pmatrix} = 0$$

Hankel system

$$\begin{pmatrix} 4 & 20 & 85 & 376 \\ 20 & 85 & 376 & 1654 \\ 85 & 376 & 1654 & 7283 \\ 376 & 1654 & 7283 & 32064 \end{pmatrix} \begin{pmatrix} 1 \\ -2 \\ -4 \\ 1 \end{pmatrix} = 0$$

Hankel system

$$\begin{pmatrix} 4 & 20 & 85 & 376 \\ 20 & 85 & 376 & 1654 \\ 85 & 376 & 1654 & 7283 \\ 376 & 1654 & 7283 & 32064 \end{pmatrix} \begin{pmatrix} 1 \\ -2 \\ -4 \\ 1 \end{pmatrix} = 0$$

Conjecture: Minimal polynomial of Dickson rank sequence is $x^3 - 4x - 2x + 1$.

Hankel system

$$\begin{pmatrix} 4 & 20 & 85 & 376 \\ 20 & 85 & 376 & 1654 \\ 85 & 376 & 1654 & 7283 \\ 376 & 1654 & 7283 & 32064 \\ 1654 & 7283 & 32064 & r_8 \end{pmatrix} \begin{pmatrix} 1 \\ -2 \\ -4 \\ 1 \end{pmatrix} = 0$$

r_8 may strengthen or disprove the conjecture.
It's computation is a challenge.

3-packing

Three bits per field element.

Thus 21 elements per 64 bit word = 2.625 elements per byte.
(unpacked - int or float - 0.25 elements per byte)

(eg. 0 010 ... 000 001 010 011)

Normalized values are $0 = 000_2, 1 = 001_2, 2 = 010_2$.

Semi-normalized values are $0 = 000_2$ or $011_2, 1 = 001_2, 2 = 010_2$.
Intermediate results carry over to the third bit (and no farther).

Semi-normalization consists in clearing the third bit per entry.

add 3-packed words

input: packed semi-normalized words x, y. output: packed semi-normalized word z.

```
mask3b = 0 001 001 001 ...
z = x + y
z = z + (( z & mask3b ) >> 2 )
```

smul - scalar-vector multiplication

input: normal field element a (eg. 0 ... 010), semi-normal packed word x.

output: $z = a^*x$

```
case a = 0: z = 0;
case a = 1: z = x;
case a = 2:
  z = x << 1
  z = z | (z & mask3b) >> 2)
```

To avoid inner loop branch, apply smul at the level of vector of packed words.

axpy ($z = ax + y$), use smul and add.

3-bitslicing

Use two bits per field element, one in each word of a 2 word pair (in corresponding bit positions).

Thus 64 elements per two 64 bit words = 4 elements per byte.

Normalized values are $0 = 00_2, 1 = 01_2, 2 = 11_2$.
(all results are normalized to these values), Boothby & Bradshaw.

eg. elements 0,1,2 are represented by first three bits of the word pair x:

```
x0 = 011....
x1 = 001....
```

3-bitslicing arithmetic

```
    x0 = 011....
    x1 = 001....
```

smul:

```
case a = 2:
    z0 = x0
    z1 = x0 xor x1
```

add: 12 bit operations (6 each for z0 and z1).

axpy: smul + add (i.e. again no special tricks)

dot product: bit-wise mul, then divide and conquer (shift, add)*
Semi-normalized values are $0 = 000_2$ or $011_2, 1 = 001_2, 2 = 010_2$.
Intermediate results carry over to the third bit (and no farther).

Semi-normalization consists in clearing the third bit per entry.

packing in mantissa of floats

Use arithmetic more, bit ops less. Less tight packing, less frequent normalization.

Emphasis to date is on dot (for matrix mul), Dumas, Fousse, Salvy.

For $n \times n$ matrix and $p = 3$, choose d such that $B = 2^{d+1} > 4n$.

$x = \sum_0^d a_i B^i$

$y = \sum_0^d b_i B^i$

$z = xy$

Then $z_d = \sum_0^d a_i b_{d-i}$.

Key point: highly tuned floating point matrix multiply can be used (BLAS) followed by normalization.

GF(3) Arithmetic Comparison (MegaFFops)					
Operation	Size	float	int	packed	pf
Vector Ops					
add	10^7	120.65	165.9	4492	
scalar mul	10^7	81.15	136.5	21008	
axpy	10^7	77.96	98.46	6165	
Matrix Ops					
mv	15000	468.7	312.5	4168	
mm	10^3	3835	350.9	3226	20k

Table 2: Speed of vector and matrix operations over GF(3), using elements that are a) stored as floats and using BLAS for mm, b) stored as ints, and c) packed.

Certification Theorem:
Given
A, an $n \times n$ matrix,
H, an $n \times b$ projection matrix, and
V, an random $n \times k$ random dense matrix,
let $B = AH$ and $C = (AH|AV)$.
(B is $n \times b - k$, C is $n \times (b)$.)

If $r = \text{rank}(B) = \text{rank}(C)$
then $r = \text{rank}(A)$
with probability of error less than $1/q^k$, where q is the cardinality of the field.

Over GF(3), with k = 13, probability of error is less than 1 in a million.

Corollary - 2 sided version

Heuristic: sum by blocks. Certificate: a few extra random cols, rows

$$\begin{pmatrix} I & I & I \\ U_1 & U_2 & U_3 \end{pmatrix} \begin{pmatrix} A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,1} & A_{3,2} & A_{3,3} \end{pmatrix} \begin{pmatrix} I & V_1 \\ I & V_2 \\ I & V_3 \end{pmatrix}$$

$$B_i = \sum_j \begin{pmatrix} A_{i,j} & A_{i,j} V_j) \end{pmatrix} // b \times b + k$$

$$B = \sum_I \begin{pmatrix} B_i \\ U_i B_i \end{pmatrix} // b + k \times b + k$$

Dickson's Hadamard difference set

G = additive group of $\mathrm{GF}(p^e) \times G\mathrm{F}(p^e)$
Difference set $\mathrm{D} = \{(a^2 + gb^{2\sigma}, 2ab) | (0,0) \neq (a,b) \in \mathrm{G}\}$
where g is a generator, σ an automorphism of $\mathrm{GF}(p^e)$.
(D is the set of non-zero squares of a semi-field multiplication on G).
Adjacency matrix:

$$a_{i,j} = \begin{cases} p-1, & \text{if } i = j, \\ 1, & \text{if } e_i - e_j \in D, \\ 0, & \text{otherwise.} \end{cases}$$

units of scale

kilo	10^3	2^{10}	\$ to attend ACA
mega	10^6	2^{20}	\$ to retire
giga	10^9	2^{30}	\$ to endow a University, cycles/sec
tera	10^{12}	2^{40}	\$ of ARRA bailout
peta	10^{15}	2^{50}	arithmetic ops limit

Big dense matrix, small rank

Case $n = 3^{14}$ and $r \sim 2^{15}$.
Storage: $(3^{14})^2$ (packed) elements ~ 6 terabytes. too much!
Time: $n^2 r \sim 0.7 \times 10^{18}$. [$10^{18}$ nanoseconds ~ 32 years.] too much!

But storage of $(2^{15})^2$ (packed 2bits/element) elements is 256 megabytes (would be 4GB at 32 bit word/element), and $(2^{15})^3$ field ops is tractable amount of time.

Idea: project to a matrix of order a little larger than r while maintaining rank. Strengthened idea: project heuristically, and certify (probabilistically).

New algorithm uses $\mathrm{O}(n^2 + r^3)$ time. In $\mathrm{O}(n^2)$ time it scans the big matrix once and generates a $b \times b$ matrix of the same rank, where b is slightly greater than r. Then in $\mathrm{O}(r^3)$ time compute the rank.

For time: $n^2 \sim 0.6 \times 2^{45}, r^3 \sim 2^{45}$.
For storage: $r^2 \sim 2^{30}$.

Actual run times:

Generate $b \times b$ block (n^2 part) - 4 days.

Compute row echelon form (r^3 part) - 3 hours.

Dickson 3^{16}

Case $n = 3^{16}$ and $r \sim 2^{17}$.

Reduction phase:

n^2 ops to produce block B of order $b \sim 2^{17}$.

$n^2 = 3^{32} \sim 0.8 \times 2^{51}$

Rank phase:

Bit-slicing: 32 elements of GF(3) per 64 bit word (= 4 elt per byte).

bit-sliced B of dimension 2^{17} occupies 2^{32} bytes.

Rank computation involves $(2^{17})^3 = 2^{51}$ field ops.

Rank computation, current limits

Large matrix, small rank Dickson problem:
Routine (with packing): $n = 2^{19}, r = 2^{15}$, time $2^{38} + 2^{45}$, memory $r^2 = 2^{30}$.

Challenge: $n = 2^{25}, r = 2^{17}$, time $2^{50} + 2^{51}$, memory 2^{34}. $2^{22}, 2^{15}$.

Large matrix, large rank (Brouwer's problem)
n^3 time, n^2 space. Limit is $n < (2^{45})^{1/3}$, if $r \sim n$.

Routine: $n = 2^{15}$, time 2^{45}, memory 2^{30}.

Small challenge: $n = 2^{17}$, time 2^{51}, memory 2^{34}.

Challenge: $n = 2^{20}$, time 2^{60}, memory 2^{40}

from Field to ...

In linbox the field F is an object of a type FIElD meeting a Field concept/interface/archetype which specifies the member functions, types, and their properties.

Field Types are template parameters to generic solutions and to matrix representations..

```
//elsewhere defined
template<class Element> Modular {
... Elt \& mul(Elt \&c, const Elt \&a, const Elt \&b); ...
};
template<class Field> DenseMatrix;
template<class Field, class Matrix> void rank(integer r, const Ma

// code using rank
typedef ... Field;
```

```
Field F(3);
DenseMatrix A(F, n, n);
... A.setEntry( i, j, F.mul(x,a,b) ) ...
rank(r, A);
```

Core LinBox Arithmetic: a Suite of Matrix Domains

```
class MatrixDomain { // BLAS-like functionality
// may encapsulate packing, delayed normalizations, parallelism
class Scalar;
class Block;
// Block::Entry may be packed word, may be unnormalized.
class Matrix;
// Matrix::Entry may be packed word, may be unnormalized.
// complexity: two matrix types suffice?
...
}

template<class MatrixDomain> DenseMatrix;
template<class MatrixDomain> void rank(integer r, const MatrixDom
```