# A Randomized Baby Steps/Giant Steps Implementation of Wiedemann's Determinant Algorithm

William J. Turner
Department of Mathematics
North Carolina State University
http://www.math.ncsu.edu/~wjturner


Joint work with Erich Kaltofen.

# Abstract

Wiedemann's determinant algorithm computes the determinant of a nonsingular matrix with arbitrarily high probability by randomly perturbing the matrix and choosing random vectors to project the powers of the matrix onto the field. This sequence of field elements is linearly generated by the minimal polynomial of the matrix, and with high probability its minimal polynomial will be the characteristic polynomial of the matrix. Through the use of Chinese Remaindering and the Berlekamp/Massey algorithm, the determinant of a dense matrix can be computed in either a Las Vegas or Monte Carlo algorithm. Furthermore, a baby steps/giant steps method allows base extensions to be used to decrease the running time of the implementation.

# Wiedemann and Berlekamp/Massey Algorithms

For $A \in \mathbb{K}^{n\times n}$ and $u, v \in \mathbb{K}^n$,
let

$$a_i = u^T A^i v \text{ for } i = 0, 1, 2, \ldots$$

and

$\mu(\lambda) =$ minimal polynomial of $\{a_i\}_{i=0}^{\infty}$.

Wiedemann: randomly precondition $A$, choose random $u, v$

Berlekamp/Massey: compute $\mu(\lambda)$

Then

$$\begin{array}{c} \deg(\mu) = n \\ \Downarrow \\ \det(\lambda I - A) = \mu(\lambda) \\ \Downarrow \\ \det(A) = (-1)^n \mu(0) \end{array} \qquad \text{and} \qquad \begin{array}{c} \mu(0) = 0 \\ \Downarrow \\ \det(A) = 0 \end{array}$$

# Baby Steps/Giant Steps: Dense Case

For $i = 0, 1, \ldots, 2n-1$ compute $a_i = u^T A^i v$ by baby steps/giant steps method:

For some $c$, let $r = \lceil \sqrt{cn} \rceil$ and $s = \lceil 2n/r \rceil$

1. (Baby Steps) For $j = 0, 1, 2, \ldots, r-1$ Do $v^{[j]} \leftarrow A^j v$;

2. $Z \leftarrow A^r$;

3. (Giant Steps) For $k = 0, 1, 2, \ldots, s$ Do $u^{[k]^T} \leftarrow u^T Z^k$;

4. For $k = 0, 1, 2, \ldots, s$ Do
   For $j = 0, 1, 2, \ldots, r-1$ Do $a_{kr+j} \leftarrow \langle u^{[k]}, v^{[j]} \rangle$;

# Modular Implementation

Idea: Compute determinant of preconditioned matrix $\det(A)$ for small primes $p_1, p_2, \ldots, p_N$ using baby steps/giant steps.

- Use bounds on for number of primes needed:
  1. $\|A^r\|_\infty \leq \|A\|_\infty^r$
  2. $|\det(A)| \leq \prod_{i=1}^{n} \|\text{column}_i(A)\|_2$ (Hadamard's bound)
- Compute $v^{[j]}, A^r \bmod p_i$ for $1 \leq i \leq N_{\text{power}}$ where

$$\prod_{i=1}^{N_{\text{power}}} p_i > 2 \cdot \|A\|_\infty^r$$

- Compute $u^{[k]}, \det(A) \bmod p_i$ for $1 \leq i \leq N_{\text{det}}$
  1. Compute $v^{[j]}, A^r \bmod p_i$ with base extension if $i > N_{\text{power}}$
  2. Keep $\det(A) \bmod p_j$ if $n = \deg(\mu)$ or $0 = \mu(0)$
  3. Stop when $\prod p_j > 2 \cdot \prod_{i=1}^{n} \|\text{column}_i(A)\|_2$
- Use Chinese remaindering to recover integer determinant.

# Garner's Rule for Chinese Remainer and Base Extension

Given:

- primes $p_i$ for $1 \leq i \leq r$
- products $P_0 = 1$ and $P_i = \prod_{j=1}^{i} p_j$ for $1 \leq i \leq r$
- $u \equiv u_i \pmod{p_i}$ for $1 \leq i \leq r$

Use precomputed constants $c_{i,j}$ for $1 \leq i < j \leq r$ such that

$$c_{i,j}\, p_i \equiv 1 \pmod{p_j}$$

to compute mixed radix representation

$$u \equiv \sum_{i=1}^{r} v_i P_{i-1} \pmod{P_r}$$

For new prime $p_{\text{new}} > \max_{1 \leq i \leq r} p_i$:

- $v_i < p_{\text{new}}$ for $1 \leq i \leq r$
- $u \bmod p_{\text{new}} = \sum_{i=1}^{r} v_i (P_{i-1} \bmod p_{\text{new}}) \bmod p_{\text{new}}$

# Probabilistic Implementations

Las Vegas: Always correct, probably fast

- Use bounds to terminate loops (as above):
  1. $\prod_{i=1}^{N_Z} p_i > 2\|A\|_\infty^r$
  2. $\prod p_{i_j} > 2\prod_i^n \|\mathrm{column}_i(A)\|_2$

Monte Carlo: Always fast, probably correct

- Terminate loops when $A^r$ and $\det(A)$ have no change
- Fewer steps when far from bounds; more when at bound

# Maple Code

```
mod_det := proc(A_init, optional arguments)
  :
  Diag := linalg[diag]( seq(rand(1..B_rnd)(), i=1..n) );
  A    := evalm(Diag &* A_init);

  r := ceil(sqrt(steps * n));
  s := ceil(2 * n / r);

  u := linalg[randvector](n, entries = rand(B_rnd));
  v[0] := linalg[randvector](n, entries = rand(B_rnd));

  for i from 1 while ((P_Z[i-1] <= 2 * B_Z) and (N_Z < Th_Z)
      and (N_det < Th_det)) do

    p := next_prime(p);
    p_Z[i] := p;
    P_Z[i] := P_Z[i-1] * p;
    v_p, Z_p := baby_steps(A, v[0], r, p);
    u_p := giant_steps(Z_p, u, s, p);
    const, P_det, N_det
     := mod_det_const(v_p, u_p, n, r, s, p, const, P_det, N_det);
```

```
    if (N_det >= Th_det) then
      break;
    end if;

    v_g, Z_g, N_Z := extend_mrs(v_g, Z_g, v_p, Z_p, p_Z, i, r, n, N_Z);

  end do;

  NN := i - 1;

  for i from i while ((P_det <= 2 * B_det) and (N_det < Th_det)) do
    p := next_prime(p);

    v_p, Z_p := mrs_to_mod(v_g, Z_g, P_Z, p, NN, r);

    u_p := giant_steps(Z_p, u, s, p);

    const, P_det, N_det
     := mod_det_const(v_p, u_p, n, r, s, p, const, P_det, N_det);
  end do;

  return (-1)^n * d / product(Diag[ii,ii], ii=1..n);

end proc:
```

# Timings

- $n \times n$ matrix of small determinant
- primes larger than $2^{30}$

| | $n$ | | | |
|---|---|---|---|---|
| | 16 | 25 | 49 | 100 |
| `linalg[det]` | 3.500 | 21.109 | 316.789 | 2159.721 |
| Normal Chinese Remaindering | 0.580 | 1.921 | 14.320 | 122.750 |
| Wiedemann with `chrem` | 5.010 | 21.880 | 294.229 | 15,484.490 |
| Wiedemann with Garner's rule | 4.649 | 21.910 | 316.570 | 17,484.130 |
| Wiedemann with base extension | 4.630 | 21.289 | 315.610 | 16,590.511 |

Demonstrations of code available upon request.

## References


[1] L. Chen, W. Eberly, E. Kaltofen, B. D. Saunders, W. J. Turner, and G. Villard. Efficient matrix preconditioners for black box linear algebra. Manuscript submitted for publication, Jan. 2000.

[2] E. Kaltofen. Computer algebra in the new century: The road ahead. Lecture at the Computer Algebra Minisymposium at the 3rd European Congress of Mathematics at Barcelona, Spain, July 2000.

[3] D. E. Knuth. *The Art of Computer Programming*, volume 2. Addison–Wesley, Reading, Massachusetts, 1997.

[4] D. Wiedemann. Solving sparse linear equations over finite fields. *IEEE Trans. Inf. Theory*, IT-32:54–62, 1986.