Implementing the tangent Graeffe root finding method

Joris van der Hoeven[1] and Michael Monagan[2]

[1]CRNS, LIX, École polytechnique, France

[2]Dept. of Mathematics, Simon Fraser University, British Columbia

Motivation: sparse polynomial interpolation.

Let $f = \sum_{i=1}^{t} a_i M_i(x_1, \ldots, x_n) \in \mathbb{Z}[x_1, \ldots, x_n]$

Problem: Interpolate f modulo a prime p from values of f.
Approach: Use Ben-Or/Tiwari [1] with a smooth prime $p = \sigma 2^k + 1$.

1. Pick $\alpha \in \mathbb{F}_p^n$ at random.
   Let $m_i = M_i(\alpha)$ and $P(z) = \prod_{i=1}^{t}(z - m_i)$.
2. Evaluate $f(\alpha_1^j, \alpha_2^j, \ldots, \alpha_n^j)$ for $0 \le j < 2t$.
3. Compute $P(z) = z^t + \ldots$ using the fast EEA .... $O(M(t) \log t)$.
4. Compute the roots m_i of $P(z)$ using CZ .... $O(M(t) \log(pt) \log t)$.
5. Using PH to compute $M_i(x_1, \ldots, x_n)$ from m_i.
6. Solve a Vandermonde system for a_i .... $O(M(t) \log t)$.

In 2015 Grenet, van der Hoeven, Lecerf, [3] Tangent Graeffe Root Finding.
Factor $P(z)$ in $O(M(t)(\log(p/s) + \log t))$ ops in $\mathbb{F}_p$ where $s \in [4t, 8t)$.

Is Tangent Graeffe faster than Cantor-Zassenhaus in practice?

Talk Outline

- The Graeffe transform
- The tangent-Graeffe (TG) algorithm
- Improving the constant by a factor of 2
- Comparison of new C implementation with Magma's CZ implementation
- Parallelization of TG
- Current work

The Graeffe Transform

Definition (1)

The Graeffe transform of $P(z) \in \mathbb{F}_p[z]$ is

$$\mathbf{G}(P) = P(z)P(-z)|_{z=\sqrt{z}}) \in \mathbb{F}_p[z]$$

Lemma

If $P(z) = \prod_{i=1}^{d}(z - \alpha_i)$ *then* $\mathbf{G}(P) = \prod_{i=1}^{d}(z - \alpha_i^2)$.

Main idea: Let $p = \sigma 2^k + 1$. Pick $r = 2^N$ such that $s = (p-1)/r \in [4d, 8d)$.
Compute $\tilde{P} = \mathbf{G}^{(N)}(P)$. Then $\tilde{P} = \prod_{i=1}^{d}(z - \alpha_i^r)$.

Let $\beta_i = \alpha_i^r$. Observe $(p-1)/r = s \Rightarrow \beta_i^s = 1$.
Pick ω with order s in $\mathbb{F}_p$ and compute $\{\omega^i : \tilde{P}(\omega^i) = 0 \leq i < s\} = \{\beta_i\}$.

The **tangent** Graeffe transform.

How do we obtain α_i from $\beta_i = \alpha_i^r$ where $r = 2^N$?

Let $\tilde{P}(z) = P(z+\epsilon) \mod \epsilon^2 \in \mathbb{F}_p[\epsilon, z]/(\epsilon^2)$.

1. $\tilde{P}(z) = P(z) + P'(z)\epsilon$
2. $\mathbf{G}(\tilde{P}(z)) = P(z)P(-z) + (P(z)P'(-z) + P(-z)P'(z))\epsilon$
3. $\mathbf{G}^{(N)}(\tilde{P}(z)) = A(z) + B(z)\epsilon$ where $A(z) = \mathbf{G}^{(N)}(P)$

Lemma

If $A(\beta) = 0$ and $A'(\beta) \neq 0$ then $\alpha = \frac{r\beta A'(\beta)}{B(\beta)}$ is a root of $P(z)$.

Compute $\mathbf{G}^{(N)}(P(z+\epsilon)) = A(z) + B(z)\epsilon$.
Compute $A(\omega^i), A'(\omega^i), B(\omega^i)$ for $0 \leq i < s$.

The Tangent Graeffe Algorithm

Input: $P \in \mathbb{F}_p[z]$ of degree d with d distinct roots in $\mathbb{F}_p$ and $p = \sigma 2^k + 1$ with $2^k > 4d$.
Output: the set $\{\alpha_1, \ldots, \alpha_d\}$ of roots of P.

1. If $d = 0$ then return ϕ.
2. Let $s \in [4d, 8d)$ such that $s | (p-1)$ and set $r := (p-1)/s = 2^N$.
3. Pick $\tau \in \mathbb{F}_p$ at random and compute $P^* := P(z+\tau) \in \mathbb{F}_p[z]$ .... $O(M(d))$.
4. Compute $\tilde{P} := P^*(z) + P^*(z)'\epsilon$. // $= P^*(z+\epsilon) \mod \epsilon^2$.
5. For $i = 1, \ldots, N$ set $\tilde{P} := \mathbf{G}(\tilde{P})(z) \mod \epsilon^2$ .... $O(NM(d))$.
6. Let ω have order s in $\mathbb{F}_p$. Let $\tilde{P}(z) = A(z) + B(z)\epsilon$.
   Evaluate $A(\omega^i), A'(\omega^i)$ and $B(\omega^i)$ for $0 \le i < s$ using Bluestein .... $3M(s) + O(s)$.
7. If $P(\tau) = 0$ then set $S := \{\tau\}$ else set $S := \phi$.
8. For $\beta \in \{1, \omega, \ldots, \omega^{(s-1)}\}$
   if $A(\beta) = 0$ and $A'(\beta) \neq 0$ set $S := S \cup \{r\beta A'(\beta)/B(\beta) + \tau\}$.
9. Compute $Q := \prod_{\alpha \in S}(z - \alpha)$ and set $R = P/Q$ .... $O(M(d) \log d)$.
10. Recursively determine the set of roots S' of R and return $S \cup S'$.

For $s \in [4d, 8d)$, on average, we get at least $e^{-1/4} = 78\%$ of the roots.
Total cost $O(NM(d) + M(d)\log d + M(s)) = O(M(d)\log(p/s) + M(d)\log d)$.

Improving the constant in $\mathbf{G}(P)$ and $\mathbf{G}^{(N)}(P)$

$$\mathbf{G}(P) = P(z)P(-z)|_{z=\sqrt{z}} \ \text{ and } \ d = \deg P$$

Proposition (1+2)

We can compute $\mathbf{G}(P)$ *in* $F(2d) + F(d) = 1/2M(d)$.
We can compute $\mathbf{G}^{(N)}(P)$ *in* $(2N+1)F(d) = (1/3N + 1/6)M(d)$.

This compares with $2/3M(d)$ and $2/3NM(d)$ in [GHL 2015].

In the FFT, if $\omega^n = 1$ and $n = 2^k$ then $\omega^{n/2+i} = -\omega^i$ so

$$\begin{aligned} FFT(P(z)) &= [P(1), P(\omega), P(\omega^2), \ldots, P(-1), P(-\omega), P(-\omega^2), \ldots] \\ FFT(P(-z)) &= [P(-1), P(-\omega), P(\omega^2), \ldots, P(1), P(\omega), f(\omega^2), \ldots] \end{aligned}$$

Also $FFT(H := P(z)P(-z))$ is

$$[H(1), H(\omega), H(\omega^2), \ldots, H(1), H(\omega), H(\omega^2), \ldots]$$

We can compute the inverse FFT with an FFT of size d.
Cost of $\mathbf{G}(P)$: $F(2d) + 0 + F^{-1}(d) < 1.5F(2d) < 1/2M(d)$.

Benchmark 1: Tangent-Graeffe v. Cantor-Zassenhaus

We implemented TG in C using the FFT for $\mathbf{G}(P)$ and for arithmetic in $\mathbb{F}_p[z]$.

Table: Sequential timings in CPU seconds for $p = 3 \cdot 29 \cdot 2^{56} + 1$ and using $s \in [2d, 4d)$.
Intel Xeon E5 2660 CPU, 8 cores, 2.2 GHz base, 3.0 GHz turbo, 64 gigabytes RAM

	Our sequential TG implementation in C						Magma CZ timings	
d	total	first	%roots	$\mathbf{G}^{(N)}$	step6	step9	V2.25-3	V2.25-5
$2^{12}-1$	0.11s	0.07s	69.8%	0.04s	0.02s	0.01s	23.22s	8.43
$2^{13}-1$	0.22s	0.14s	69.8%	0.09s	0.03s	0.01s	56.58s	18.94
$2^{14}-1$	0.48s	0.31s	68.8%	0.18s	0.07s	0.02s	140.76s	44.07
$2^{15}-1$	1.00s	0.64s	69.2%	0.38s	0.16s	0.04s	372.22s	103.5
$2^{16}-1$	2.11s	1.36s	68.9%	0.78s	0.35s	0.10s	1494.0s	234.2
$2^{17}-1$	4.40s	2.85s	69.2%	1.62s	0.74s	0.23s	6108.8s	534.5
$2^{18}-1$	9.16s	5.91s	69.2%	3.33s	1.53s	0.51s	NA	1219.
$2^{19}-1$	19.2s	12.4s	69.2%	6.86s	3.25s	1.13s	NA	2809.
$2^{20}-1$	39.7s	25.7s	69.2%	14.1s	6.77s	2.46s	NA	6428.

Conclusion: TG is a lot (100 times) faster than CZ.

Benchmark 2: Parallelizing Tangent-Graeffe in Cilk C

Using Cilk C, we parallelized the underlying FFT, and $\mathbf{G}^{(N)}$ in step 5 and the product $Q = \prod_{\alpha \in S}(z - \alpha)$ in step 9.

Table: Real times in seconds for 1 core (8 cores) and $p = 3 \cdot 29 \cdot 2^{56} + 1$.

d	total		first	$\mathbf{G}^{(N)}$		step5		step9	
$2^{19}-1$	18.30s		11.98s	6.64s		3.13s		1.09s	
8 cores	9.616s	1.9x	2.938s	1.56s	4.3x	0.49s	6.4x	0.29s	3.8x
$2^{20}-1$	38.69s		25.02s	13.7s		6.62s		2.40s	
8 cores	12.40s	3.1x	5.638s	3.03s	4.5x	1.04s	6.4x	0.36s	6.7x
$2^{21}-1$	79.63s		52.00s	28.1s		13.9s		5.32s	
8 cores	20.16s	3.9x	11.52s	5.99s	4.7x	2.15s	6.5x	0.85s	6.3x
$2^{22}-1$	166.9s		107.8s	57.6s		28.9s		11.7s	
8 cores	41.62s	4.0x	23.25s	11.8s	4.9x	4.57s	6.3x	1.71s	6.8x
$2^{23}-1$	346.0s		223.4s	117.s		60.3s		25.6s	
8 cores	76.64s	4.5x	46.94s	23.2s	5.0x	9.45s	6.4x	3.54s	7.2x
$2^{24}-1$	712.7s		459.8s	238.s		125.s		55.8s	
8 cores	155.0s	4.6x	95.93s	46.7s	5.1x	19.17	6.5x	7.88s	7.1x
$2^{25}-1$	1465.s		945.0s	481.s		259.s		121.s	
8 cores	307.7s	4.8x	194.6s	92.9s	5.2x	39.2s	6.6x	16.9s	7.2x

Current work

Can we factor $P(z) = z^{10^9} + \ldots$ in $\mathbb{F}_p[z]$ for $p = 5 \cdot 2^{55} + 1$?
Note: we need 8 gigabytes for the input and 8 gigabytes for the output.

Current work

Can we factor $P(z) = z^{10^9} + \ldots$ in $\mathbb{F}_p[z]$ for $p = 5 \cdot 2^{55} + 1$?
Note: we need 8 gigabytes for the input and 8 gigabytes for the output.

Yes! time = 4000s, space = 121 GB
Used an Intel E5 2680 CPU with 10 cores and 128 GB RAM.

Current work

Can we factor $P(z) = z^{10^9} + \dots$ in $\mathbb{F}_p[z]$ for $p = 5 \cdot 2^{55} + 1$?
Note: we need 8 gigabytes for the input and 8 gigabytes for the output.

Yes! time $=$ 4000s, space $=$ 121 GB
Used an Intel E5 2680 CPU with 10 cores and 128 GB RAM.

To evaluate $A(\omega^i), A'(\omega^i), B(\omega^i)$ for $0 \le i < s = 52^{30}$
Space: $3s + 3n = 504 GB$ with $n = 2^k > 2s$ for $M(s)$ using Bluestein.

Use $s \in [2d, 4d)$ instead of $s \in [4d, 8d)$.
For $s = 5 \cdot 2^{29}$, a DFT$(5 \cdot 2^{29})$ can be done using $5F(2^{29}) + 2^{29}F(5) + O(s)$.
Space: $3s + 1.2s = 84 GB$.

Current work cont.

Tangent-Graeffe cost for $s \in [\lambda d, 2\lambda d)$.

$\mathbf{G}^{(N)}(P)$	$Q := \prod_{\alpha \in S} (z - \alpha)$
$< \frac{1}{3} e^{1/\lambda} M(d) \log_2 \frac{p}{\lambda d} + \ldots$	$< \frac{1}{4} M(d) \log_2 d + \ldots$

Cantor-Zassenhaus cost

$h := (z + \alpha)^{(p-1)/2} \bmod P(z)$	$g := \gcd(h(z) - 1, P(z))$
$< \frac{7}{6} M(d) \log \frac{p}{2d} \log_2 d + \ldots$	$< \frac{5}{12} M(d) \log_2^2 d + \ldots$

For HalfGcd, MCA Th. 11.10 gives the bound $10 M(d) \log_2^2 d + O(M(d))$ for Algorithm 11.6 Half gcd [2].

References

M. Ben-Or and P. Tiwari.
A deterministic algorithm for sparse multivariate polynomial interpolation.
In Proceedings of STOC '88, ACM Press, pp. 301–309, 1988.

J. von zur Gathen and J. Gerhard.
Modern Computer Algebra.
3rd ed., Cambridge University Press, 2013.

Bruno Grenet, Joris van der Hoeven and Gregoire Lecerf.
Randomized Root Finding over Finite FFT-fields using Tangent Graeffe Transforms.
In Proceedings of ISSAC 2015, pp. 197–204, ACM, 2015.

Joris van der Hoven and Michael Monagan.
Implementing the tangent Graeffe root finding algorithm.
To appear in Proceedings of ICMS 2020, LNCS **12097**, 2020.
Preprint available on the HAL achive.