

Fast resolution of differential equations and applications

Alin Bostan

ALGO Project, INRIA Rocquencourt, France

Context

devise *fast* algorithms for solving (in power series) ordinary differential equations with power series coefficients

applications: cryptography, computer algebra, combinatorics, control theory, algebraic complexity

- ▶ *fast* means using *few operations* (\pm, \times, \div) in the base field \mathbb{K} .

More precisely

Given a linear differential equation with coefficients a_i in $\mathbb{K}[[t]]$

$$a_r(t)y^{(r)}(t) + \dots + a_1(t)y'(t) + a_0(t)y(t) = 0$$

compute **the first N terms** of a basis of power series solutions.

Same problem for general $r \times r$ linear systems $Y' = AY$.

- ▶ Naive algorithm (undetermined coefficients), $\mathcal{O}(r^2 N^2)$ ops. in \mathbb{K} .
- ▶ Best that can be hoped: complexity **linear** in N and **quadratic** in r .

Fast polynomial (matrix) multiplication

$$\begin{aligned} \mathbf{M}(N) &= \text{complexity of polynomial multiplication in degree } < N \\ &= \mathcal{O}(N^2) \text{ by the naive algorithm} \\ &= \mathcal{O}(N^{1.58}) \text{ by Karatsuba's algorithm} \\ &= \mathcal{O}(N^{\log_{\alpha}(2\alpha-1)}) \text{ by the Toom-Cook algorithm} \\ &= \mathcal{O}(N \log N \log \log N) \text{ by the Schönhage--Strassen FFT} \end{aligned}$$

$$\begin{aligned} \mathbf{MM}(r, N) &= \text{complexity of polynomial matrix mult, size } r, \text{ deg } < N \\ &= \mathcal{O}(r^{\omega} \mathbf{M}(N)) \text{ by the Cantor--Kaltofen algorithm} \\ &= \mathcal{O}(r^{\omega} N + r^2 \mathbf{M}(N)) \text{ by the B.--Schost algorithm} \end{aligned}$$

Previous results for $Y' = AY$

- $r = 1$ [Brent (1975)] * reduction to exponentials of power series + Newton iteration for exponential, $\mathcal{O}(M(N))$.
- $r > 1$ [Brent & Kung (1978)] ** cascade order reduction via generalized Riccati equations + linearization, $\mathcal{O}(r^r M(N))$.
- $r > 1$ [van der Hoeven (2002)] divide-and-conquer algorithm, $\mathcal{O}(r^2 M(N) \log N)$.

* Multiple-precision zero-finding methods and the complexity of elementary function evaluation, *Analytic computational complexity*.

** Fast algorithms for manipulating formal power series, *Journal of the ACM*.

New results

[B, Chyzak, Ollivier, Salvy, Schost, Sedoglavic] \star solve $Y' = AY$, where $A \in \mathcal{M}_{r \times r}(\mathbb{K}[[t]] <_N)$, using $\mathcal{O}(\mathbf{MM}(r, N))$ operations in \mathbb{K} , by a **Newton-type iteration**.

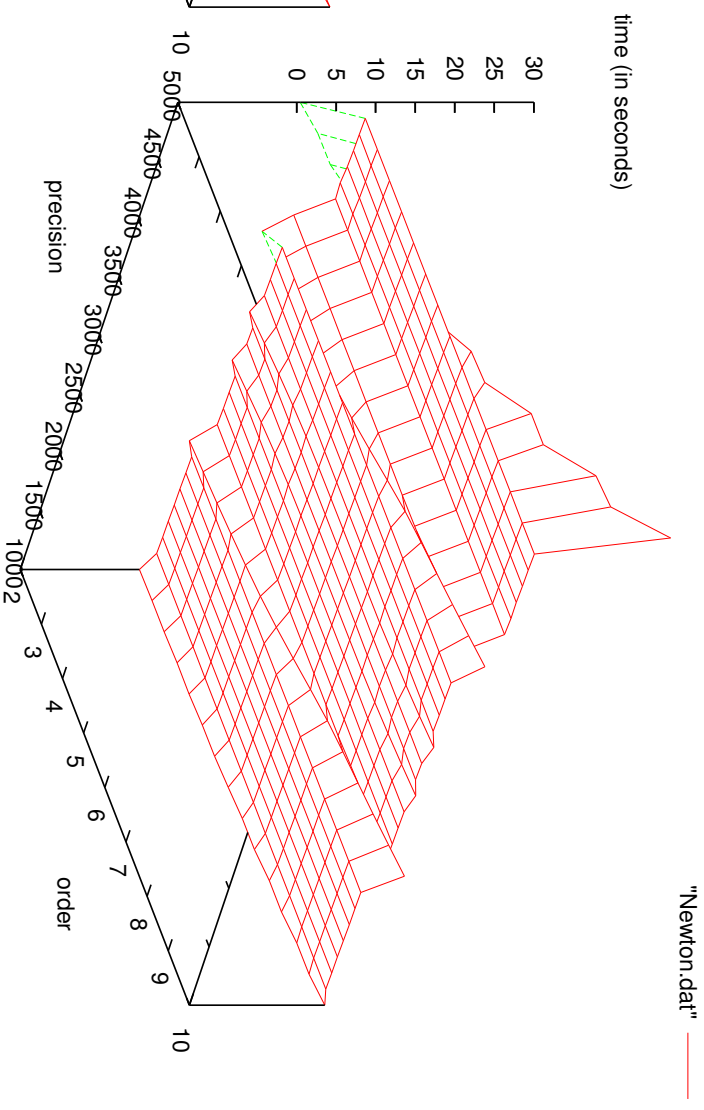
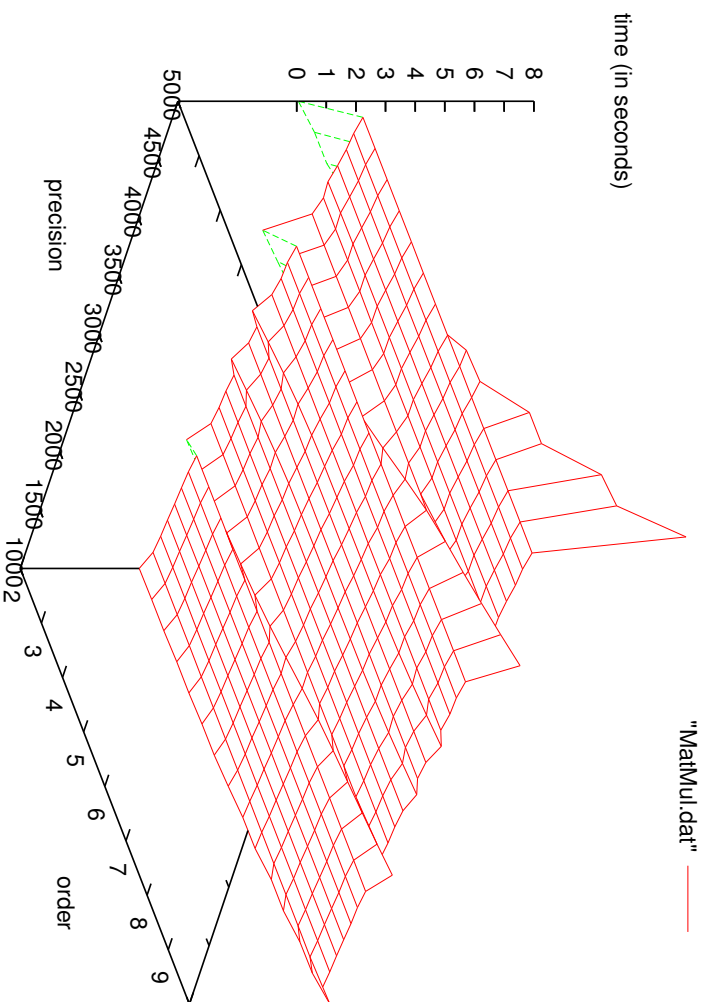
[B, González-Vega, Perdry, Schost] \star , [B, Schost] $\star\star$ extend this approach to the small characteristic case.

[B, Flajolet, Salvy, Schost] \star nearly optimal computation of bivariate resultants for algebraic numbers.

[B, Morain, Salvy, Schost] \star nearly optimal computation of morphisms between elliptic curves.

-
- \star Fast computation of power series solutions of systems of differential equations, *preprint*.
 - \star From Newton sums to coefficients, complexity issues in characteristic p , *MEGA'05*.
 - $\star\star$ Fast algorithms for p -adic differential equations, *preprint*.
 - \star Fast computation of special resultants, *Journal of Symbolic Computation*, 2005.
 - \star Fast algorithms for computing isogenies between elliptic curves, *preprint*.

Experimental results, $Y' = AY$



Polynomial matrix multiplication vs. solving $Y' = AY$ by our new method.

Experimental results, $Y' = AY$

$N \cdot r$	2	4	8	16
256	0.02 vs. 2	0.08 vs. 6	0.44 vs. 28	3 vs. 169
512	0.04 vs. 8	0.17 vs. 25	1 vs. 113	6.41 vs. 688
1024	0.08 vs. 32	0.39 vs. 104	2.30 vs. 484	15 vs. 2795
2048	0.18 vs. 128	0.94 vs. 424	5.54 vs. 2025	36 vs. > 3h *
4096	0.42 vs. 503	2.26 vs. 1686	13.69 vs. 8348	92 vs. > 1/2 day*

Timings (in sec.) for $r \times r$ systems at precision N : new vs. naive

Newton's tangent method: power series case

Let $\varphi : \mathbb{K}[[t]] \rightarrow \mathbb{K}[[t]]$. To solve $\varphi(g) = 0$ in $\mathbb{K}[[t]]$, iterate

$$g_{\kappa+1} = g_{\kappa} - \frac{\varphi(g_{\kappa})}{\varphi'(g_{\kappa})} \pmod{t^{2^{\kappa+1}}}.$$

- ▶ The number of correct coefficients **doubles** after each iteration.
- ▶ **Total cost** = $2 \times$ (the cost of the **last** iteration).

Theorem [Cook (1966), Sieveking (1972) & Kung (1974), Brent (1975)]
Division, logarithm and exponential of power series in $\mathbb{K}[[t]]$ can be computed at precision N using $\mathcal{O}(M(N))$ operations in \mathbb{K} .

Division and logarithm of power series

- ▶ To compute the reciprocal of $f \in \mathbb{K}[[t]]$, choose $\varphi(g) = 1/g - f$:

$$g_0 = \frac{1}{f_0} \quad \text{and} \quad g_{\kappa+1} = 2g_{\kappa} - fg_{\kappa}^2 \quad \text{mod } t^{2^{\kappa+1}} \quad \text{for } \kappa \geq 0.$$

- ▶ division of power series at precision N in $\mathcal{O}(\mathbf{M}(N))$.
- ▶ $\log(f) = -\sum_{i \geq 1} \frac{(1-f)^i}{i}$ of $f \in 1 + t\mathbb{K}[[t]]$ in $\mathcal{O}(\mathbf{M}(N))$ by:
 1. computing the Taylor expansion of $h = f'/f$ modulo t^{N-1} ;
 2. taking the antiderivative of h .

Exponentials of power series

- ▶ To compute the first N terms of the exponential $\exp(f) = \sum_{i \geq 0} \frac{f^i}{i!}$, choose $\varphi(g) = \log(g) - f$. Iteration

$$g_0 = 1 \quad \text{and} \quad g_{\kappa+1} = g_{\kappa} - g_{\kappa} (\log(g_{\kappa}) - f) \quad \text{mod } t^{2^{\kappa+1}} \quad \text{for } \kappa \geq 0.$$

- ▶ First order linear differential equations: $a f' + b f = c$.
 - if $c = 0$ then the solution is $f_0 = \exp(-\int b/a)$
 - else, variation of constants: $f = f_0 g$, where $g' = c/a f_0$.
- ▶ main difficulty for higher orders: for non-commutativity reasons, the matrix exponential $Y(t) = \exp(\int A(t))$ is not a solution of $Y' = A(t)Y$.

Outline of the rest of the talk

- (I) application of **fast exp** to manipulations with algebraic numbers
- (II) non-linear equations of 1st order
 - 1. the **Brent & Kung algorithm**
 - 2. our application to **fast isogeny computation**
- (III) equations of arbitrary orders
 - 1. the **Brent & Kung cascade algorithm**
 - 2. our **new Newton-type algorithm**

Application: conversion coefficients \leftrightarrow power sums

Any polynomial $F = t^N + A_1 t^{N-1} + \dots + A_N$ in $\mathbb{K}[t]$ can be represented by its first N power sums $S_i = \sum_{F(x)=0} x^i$.

Conversions coefficients \leftrightarrow power sums can be performed

- either in $\mathcal{O}(N^2)$ using **Newton identities** (naive way):

$$iA_i + S_1 A_{i-1} + \dots + S_i = 0, \quad 1 \leq i \leq N.$$

- or in $\mathcal{O}(M(N))$ using **generating series** [Schönhage, 1982]:

$$\frac{\text{rev}(F)'}{\text{rev}(F)} = - \sum_{i \geq 0} S_{i+1} t^i \quad \text{and} \quad \text{rev}(F) = \exp \left(- \sum_{i \geq 1} \frac{S_i}{i} t^i \right).$$

Application: special bivariate resultants

Composed products and sums: manipulation of algebraic numbers

$$(F, G) \mapsto F \otimes G := \prod_{F(\alpha)=0, G(\beta)=0} (t - \alpha\beta), \quad \text{of degree } N$$

Naive approach, linear algebra. $\mathcal{O}(N^\omega)$ ops. in \mathbb{K} .

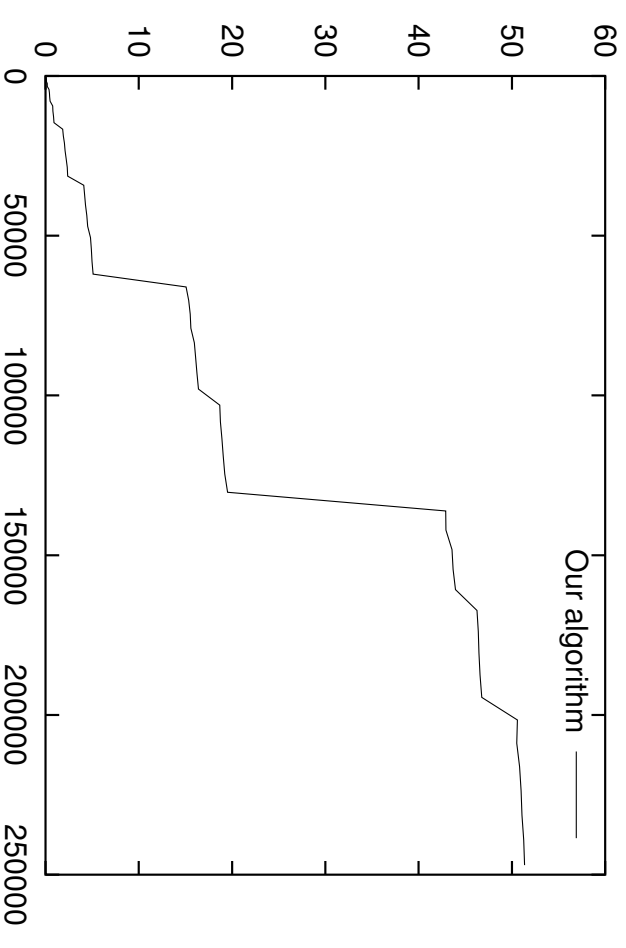
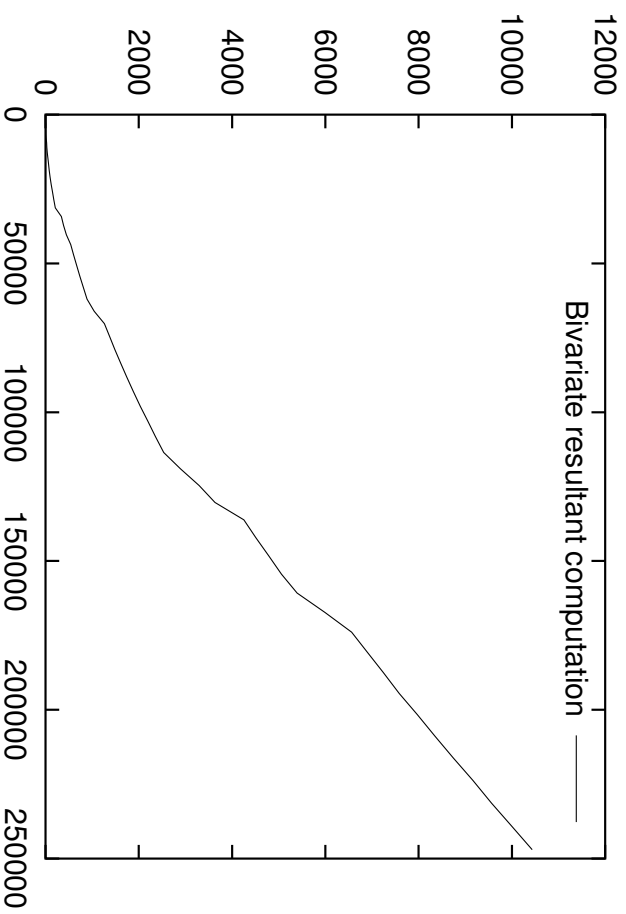
Bivariate resultant. $\text{Res}_x(x^{\deg(F)}F(t/x), G(x))$, $\mathcal{O}(N^{1.5})$ ops. in \mathbb{K} .

B-Flajolet-Salvy-Schost. Idea: \otimes is trivial in Newton representation,

$$\text{since } \left(\sum_{\alpha} \alpha^s\right) \left(\sum_{\beta} \beta^s\right) = \sum_{\alpha, \beta} (\alpha\beta)^s, \quad \mathcal{O}(M(N)) \text{ ops. in } \mathbb{K}.$$

Applications (crypto): composed products over finite fields, with input degrees up to 500 or 1000, so that $N > 200000$ can be expected.

Timings: comparison with bivariate resultant



timings in seconds vs. output degree N , $\mathbb{K} = \mathbb{F}_p$, 26 bits prime p

1st order nonlinear differential equations [Brent & Kung, 1978]

Let $G \in \mathbb{K}[[t]][[z]]$, $\alpha, \beta \in \mathbb{K}$. Compute (the first N terms) of $f \in \mathbb{K}[[t]]$

$$f'^2 = G(t, f), \quad f(0) = \alpha, \quad f'(0) = \beta.$$

Idea: Supposing that $f_1 = f \bmod t^s$ is known, deduce $f \bmod t^{2s-1}$.

Write $f = f_1 + f_2 \bmod t^{2s-1}$; then f_2 satisfies the **linearized equation**

$$2f_1'f_2' - G_z(t, f_1)f_2 = G(t, f_1) - f_1'^2 \bmod t^{2s-2}$$

↪ doubling the nb. of correct coefficients of the solution f in $\mathcal{O}(\mathbf{M}(s))$.

↪ $f \bmod t^N$ can be computed using $\mathcal{O}(\mathbf{M}(N))$ operations.

Application: isogeny computation

Isogeny between elliptic curves = rational map + group homomorphism.

Basic example: multiplication-by- ℓ map $[\ell] : P \mapsto P + \dots + P,$

$$[\ell](x, y) = \left(\frac{\phi_\ell(x, y)}{\psi_\ell(x, y)^2}, \frac{\omega_\ell(x, y)}{\psi_\ell(x, y)^3} \right).$$

The **division polynomials** ψ_ℓ , of degree $\Theta(\ell^2)$, play a crucial role in Schoof's algorithm for point counting.

Atkin & Elkies (SEA algo): work with a divisor f_ℓ of degree $\Theta(\ell)$ of ψ_ℓ ; f_ℓ is determined by computing a degree ℓ isogeny \mathcal{I}_ℓ . Cost $\mathcal{O}(\ell^2)$.

Our contribution: an algorithm for \mathcal{I}_ℓ of **complexity quasi-linear** in ℓ .

Idea: $\mathcal{I}_\ell(x, y) = \left(\frac{N(x)}{D(x)}, y \left(\frac{N(x)}{D(x)} \right)' \right)$, thus $u = N/D$ satisfies

$$(x^3 + Ax + B)u'^2 = u^3 + \tilde{A}u + \tilde{B}.$$

Example: the 9th division polynomial of $E : y^2 = x^3 + x$ over \mathbb{F}_{97} is

$$\begin{aligned} \psi_9 = & 9x^{40} + 74x^{38} + x^{36} + 83x^{34} + 51x^{32} + 20x^{30} + 49x^{28} + 29x^{26} + 50x^{24} + 17x^{22} + \\ & 69x^{20} + 52x^{18} + 17x^{16} + 94x^{14} + 47x^{12} + 89x^{10} + 27x^8 + 47x^6 + 38x^4 + 42x^2 + 1 \end{aligned}$$

Instead, compute an isogeny from E to $\tilde{E} : y^2 = x^3 + 81x$, by solving

$$(x^3 + x)u'^2 = u^3 + 81u. \quad \text{Power series solution} \quad u(x) = 81x + 35x^3 + 6x^5 + 72x^7 + 62x^9 + 90x^{11} + 66x^{13} + 62x^{15} + 10x^{17} + \mathcal{O}(x^{19})$$

$$\mathcal{I}_9 : (x, y) \mapsto \left(\frac{x^9 + 85x^7 + 30x^5 + 36x^3 + 9x}{(x^4 + 2x^2 + 32)^2}, y u'(x) \right)$$

The divisor f_e of ψ_e can be recovered from u in $\mathcal{O}(\mathbf{M}(\ell))$ ops. in \mathbb{K} .

Brent & Kung's algorithm for 2nd order equations

1. reduce $y''(t) + a(t)y'(t) + b(t)y(t) = 0$ to first order equations

▶ factor $D^2 + a(t)D + b(t)$ as $(D + S(t))(D + T(t))$:

$$S + T = a, T' + ST = b, \quad \text{thus} \quad T' + aT - T^2 - b = 0$$

2. reduce this **Riccati equation** to a linear equation (by linearization)

3. find S, T and solve two linear 1st order equations to get $y(t)$

▶ generalizes to arbitrary orders:

$$\text{Lin}(r, N) = \text{NonLin}(r - 1, N) + \mathcal{O}(r \mathbf{M}(N)) + \text{Lin}(r - 1, N),$$

$$\text{NonLin}(r - 1, N) = \mathcal{O}(\text{Lin}(r - 1, N))$$

$$\hookrightarrow \text{Lin}(r, N) = \mathcal{O}(r^r \mathbf{M}(N)).$$

Newton iteration hits again

Suppose we have to solve a “functional” equation $\phi(Y) = 0$, where

$$\phi : \mathcal{M}_{r \times r}(\mathbb{K}[[t]]) \rightarrow \mathcal{M}_{r \times r}(\mathbb{K}[[t]]) \text{ is differentiable.}$$

Define the sequence $Y_{\kappa+1} = Y_{\kappa} - U_{\kappa+1}$, where

- $U_{\kappa+1}$ is a solution of valuation $\geq 2^{\kappa+1}$ of the linearized equation
$$D\phi|_{Y_{\kappa}} \cdot U = \phi(Y_{\kappa}),$$
- $D\phi|_{Y_{\kappa}}$ is the differential of ϕ at Y_{κ} .

Then, the sequence Y_{κ} converges quadratically to the solution Y .

Application: matrix inversion

To compute the inverse Z of a matrix Y of power series:

- choose the map $\phi : Z \mapsto I - YZ$ with differential $Z \mapsto -YZ$
- the equation for U becomes $-YU = I - YZ_{\kappa} \pmod{t^{2^{\kappa+1}}}$
- solution $U = -Y^{-1}(I - YZ_{\kappa}) = -Z_{\kappa}(I - YZ_{\kappa}) \pmod{t^{2^{\kappa+1}}}$

This yields the Newton–Schulz iteration for Y^{-1} [Schulz, 1933]

$$Z_{\kappa+1} = Z_{\kappa} + Z_{\kappa}(I_r - YZ_{\kappa}) \pmod{t^{2^{\kappa+1}}}.$$

$$\mathbf{C}_{\text{inv}}(N) = \mathbf{C}_{\text{inv}}(N/2) + \mathcal{O}(\mathbf{M}(r, N)) \rightsquigarrow \mathbf{C}_{\text{inv}}(N) = \mathcal{O}(\mathbf{M}(r, N))$$

Solving arbitrary order differential equations

To compute the solution Y of the system $Y' = AY$

- choose the map $\phi : Y \mapsto Y' - AY$, with differential ϕ .
- the equation for U is $U' - AU = Y'_\kappa - AY_\kappa \pmod{t^{2^{\kappa+1}}}$
- the method of variation of constants yields the solution

$$U = Y_\kappa V_\kappa \pmod{t^{2^{\kappa+1}}}, \quad Y'_\kappa - AY_\kappa = Y_\kappa V'_\kappa \pmod{t^{2^{\kappa+1}}}.$$

This yields the BCOSSS iteration for Y :

$$Y_{\kappa+1} = Y_\kappa - Y_\kappa \int Y_\kappa^{-1} (Y'_\kappa - AY_\kappa) \pmod{t^{2^{\kappa+1}}}.$$

$$\mathbf{C}_{\text{solve}}(N) = \mathbf{C}_{\text{solve}}(N/2) + \mathcal{O}(\mathbf{M}(r, N)) \quad \mapsto \quad \mathbf{C}_{\text{solve}}(N) = \mathcal{O}(\mathbf{M}(r, N))$$