

**Fast computation of power series solutions
of systems of differential equations**

Alin Bostan

ALGO, INRIA Rocquencourt

joint work with

F. Chyzak, F. Ollivier,

B. Salvy, É. Schost, A. Sedoglavic

Context

find *fast* algorithms for solving (in power series) ordinary differential equations

applications: combinatorics, numerical analysis (control), algebraic complexity

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

More precisely

Given a linear differential equation with power series coefficients

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

compute the first N terms of a (basis of) power(s) series solution(s) $y(t)$.

- ▶ naive algorithm (undetermined coefficients), $\mathcal{O}(rN^2)$.
- ▶ Best that can be hoped: complexity linear in N and polynomial in r .
- ▶ Same problem for linear systems $Y' = AY$ and for non-linear systems.

Fast polynomial (matrix) multiplication

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

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

Previous results

- $r = 1$ Brent & Kung (1978): reduction to exponentials of power series + Newton iteration, $\mathcal{O}(\mathbf{M}(N))$.
- $r = 2$ Brent & Kung (1978): reduction to Riccati non-linear equations + linearization, $\mathcal{O}(\mathbf{M}(N))$.
- $r > 1$ Brent & Kung (1978), van der Hoeven (2002): order reduction + linearization, $\mathcal{O}(r^r \mathbf{M}(N))$.
- $r > 1$ van der Hoeven (2002) 1 solution in $\mathcal{O}(r \mathbf{M}(N) \log N)$, model of relaxed multiplication

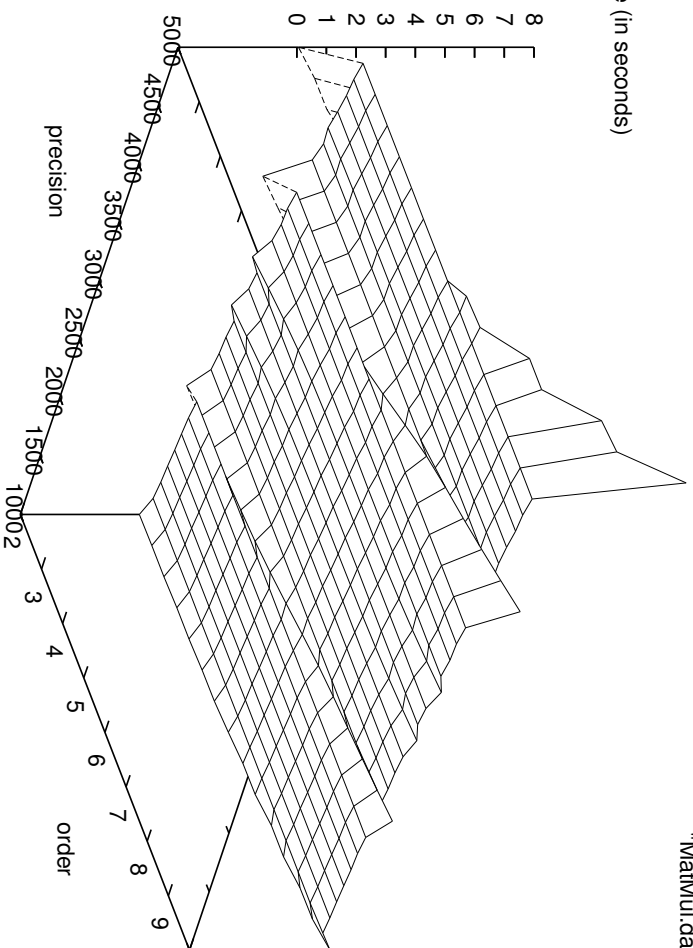
New results

Problem (input, output)	constant coefficients	polynomial coefficients	power series coefficients	output size
(eqn, basis)	$\mathcal{O}(M(r)N)$	$\mathcal{O}(dr^2N)$	$\mathcal{O}(MM(r, N))$	$\mathcal{O}(rN)$
(eqn, 1 sol)	$\mathcal{O}(M(r)N/r)$	$\mathcal{O}(drN)$	$\mathcal{O}(rM(N)\log N)$	$\mathcal{O}(N)$
(sys, basis)	$\mathcal{O}(r^M(r)N)$	$\mathcal{O}(dr^\omega N)$	$\mathcal{O}(MM(r, N))$	$\mathcal{O}(r^2N)$
(sys, 1 sol)	$\mathcal{O}(M(r)N)$	$\mathcal{O}(dr^2N)$	$\mathcal{O}(r^2M(N)\log N)$	$\mathcal{O}(rN)$

Experimental results

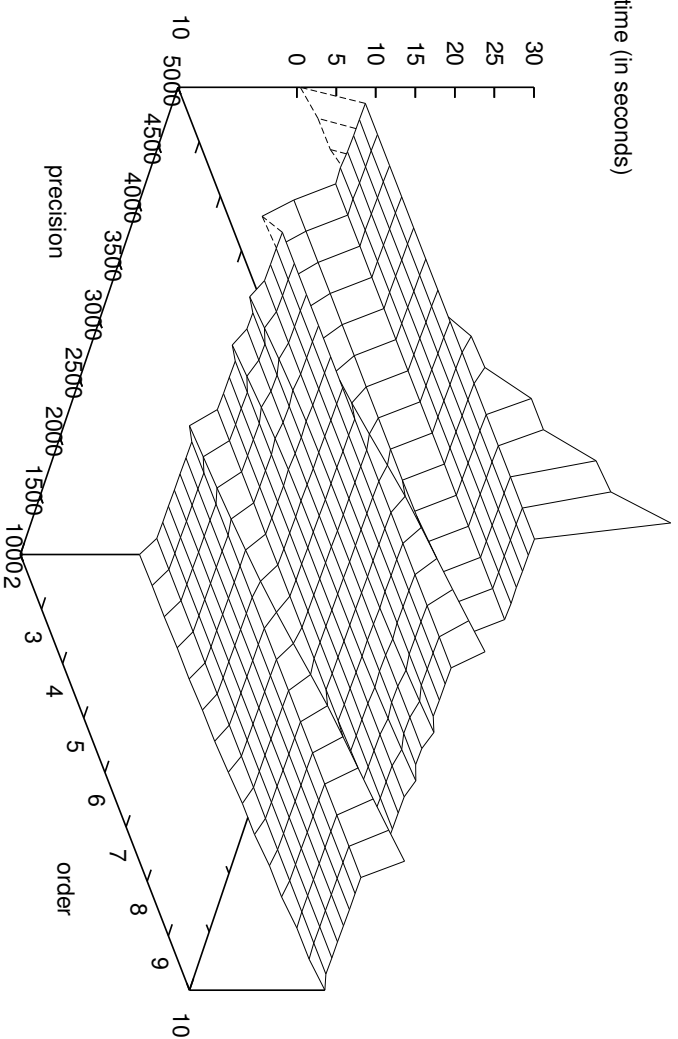
"MathMul.dat" —

time (in seconds)



"Newton.dat" —

time (in seconds)

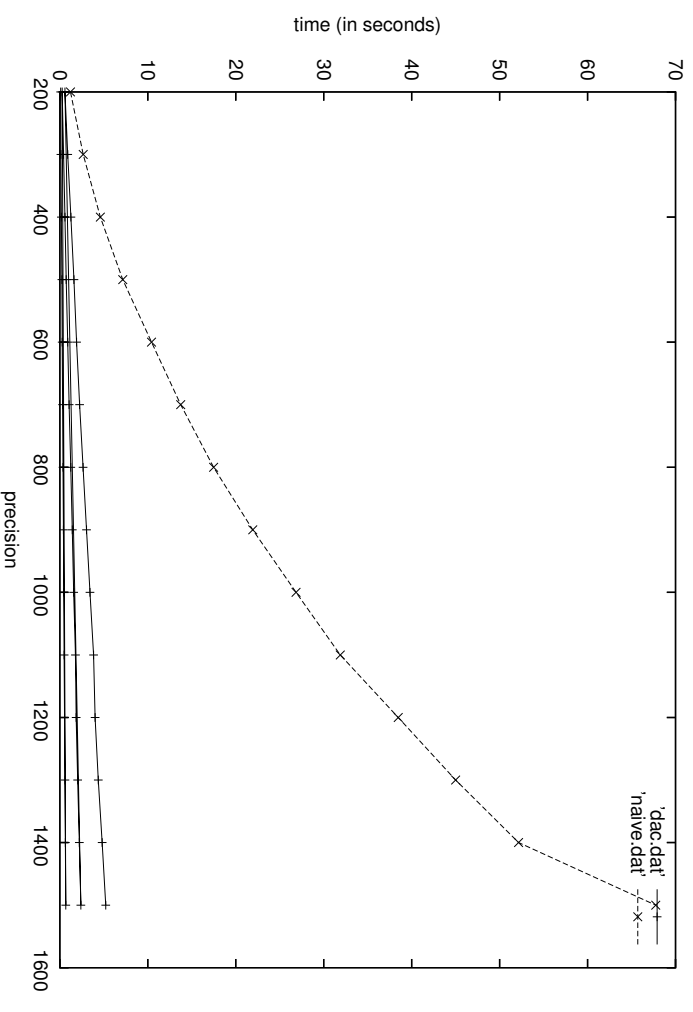
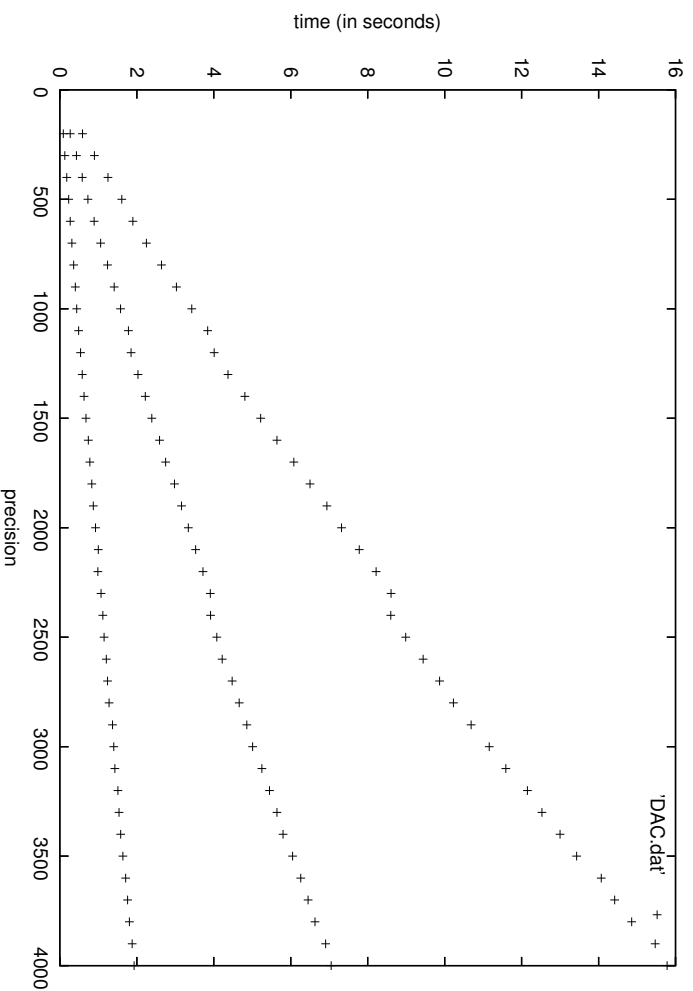


Experimental results

$N \cdot r$	2	4	8	16
256	0.02 vs. 2.09	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*

Basis, system with r equations, precision N : new vs. naive

Experimental results



Left: DAC computation of one solution for LDE of orders 2, 4, and 8

Right: DAC vs. naive, one solution of a LDE of order 2

The divide-and-conquer algorithm

Problem: solve $\mathcal{L}y = 0$, where $\mathcal{L} = \sum_i a_i(t)\delta^i$, with $\delta = t \frac{d}{dt}$.

Idea: the lowest terms of $y(t)$ only depend on the lowest terms of a_i .

Proof: if $y = y_0 + t^m y_1$, then $\mathcal{L}(\delta)y = \mathcal{L}(\delta)y_0 + t^m \mathcal{L}(\delta + m)y_1$

The divide-and-conquer algorithm

Problem: solve $\mathcal{L}y = 0$, where $\mathcal{L} = \sum_i a_i(t)\delta^i$, with $\delta = t \frac{d}{dt}$.

Idea: the lowest terms of $y(t)$ only depend on the lowest terms of a_i .

Proof: if $y = y_0 + t^m y_1$, then $\mathcal{L}(\delta)y = \mathcal{L}(\delta)y_0 + t^m \mathcal{L}(\delta + m)y_1$

DAC algorithm to solve $\mathcal{L}(\delta)y = 0 \pmod{t^{2m}}$:

1. determine y_0 , by recursively solving $\mathcal{L}(\delta)y_0 = 0 \pmod{t^m}$;
2. compute the “error” R , such that $\mathcal{L}(\delta)y_0 = t^m R \pmod{t^{2m}}$;
3. compute y_1 , by recursively solving $\mathcal{L}(\delta + m)y_1 = -R \pmod{t^m}$.

$$\mathbf{C}(N) = 2\mathbf{C}(N/2) + \mathcal{O}(r\mathbf{M}(N)) \quad \blacktriangleright \quad \mathbf{C}(N) = \mathcal{O}(r\mathbf{M}(N)\log N)$$

The divide-and-conquer algorithm

Problem: solve $\mathcal{L}y = 0$, where $\mathcal{L} = \sum_i a_i(t)\delta^i$, with $\delta = t \frac{d}{dt}$.

Idea: the lowest terms of $y(t)$ only depend on the lowest terms of a_i .

Proof: if $y = y_0 + t^m y_1$, then $\mathcal{L}(\delta)y = \mathcal{L}(\delta)y_0 + t^m \mathcal{L}(\delta + m)y_1$

DAC algorithm to solve $\mathcal{L}(\delta)y = 0 \pmod{t^{2m}}$:

1. determine y_0 , by recursively solving $\mathcal{L}(\delta)y_0 = 0 \pmod{t^m}$;
2. compute the “error” R , such that $\mathcal{L}(\delta)y_0 = t^m R \pmod{t^{2m}}$;
3. compute y_1 , by recursively solving $\mathcal{L}(\delta + m)y_1 = -R \pmod{t^m}$.

$$\mathbf{C}(N) = 2\mathbf{C}(N/2) + \mathcal{O}(r\mathbf{M}(N)) \quad \hookrightarrow \quad \mathbf{C}(N) = \mathcal{O}(r\mathbf{M}(N)\log N)$$

► **Newton method**: computing a *whole basis of solutions* in 1. will allow to determine y_1 in 3. from y_0 and R alone, *without a second recursive call*:

$$\tilde{\mathbf{C}}(N) = \tilde{\mathbf{C}}(N/2) + \mathcal{O}(\mathbf{M}(r, N)) \quad \hookrightarrow \quad \tilde{\mathbf{C}}(N) = \mathcal{O}(\mathbf{M}(r, N))$$

Newton iteration: power series case

Let $\varphi : \mathbb{K}[[t]] \rightarrow \mathbb{K}[[t]]$. To solve $\varphi(g) = 0$ in $\mathbb{K}[[t]]$, one can apply Newton's tangent method:

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

- ▶ 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 a 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 $\mathcal{O}(\mathbf{M}(N))$.
- ▶ $\log(f) = -\sum_{i \geq 1} \frac{1}{i} (1 - f)^i$ 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 $\exp(f) = \sum_{i \geq 0} \frac{1}{i!} f^i$, 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 differential equations:** compute the first N terms of $f \in \mathbb{K}[[t]]$ such that $a f' + b f = c$.
- if $c = 0$ then the solution is $f_0 = \exp(-\int b/a)$
 - else, *variation of constants:* $f = f_0 g$, where $g' = c/a f_0$.

Intermezzo: constant coefficients case

Problem: solve $y'(t) = A y(t)$, $y(0) = v$, where A is a scalar matrix.

Equivalent question: compute $y(t) = \exp(\int Av)$.

Warning: this equivalence is no longer true in the non-scalar case!

Idea: the Laplace transform $z_N = \sum_{i=0}^{N-1} A^i v t^i$ of $y(t)$ is the truncation at order N of $z = \sum_{i \geq 0} A^i v t^i = (I - tA)^{-1}v$.

Algorithm (sketch):

1. compute z as a rational function of degree $\leq r$ (indep. on N);
2. deduce its Taylor expansion modulo t^N : $\mathcal{O}(N/r \mathbf{M}(r))$ to expand each coordinate of z .

$$\frac{a}{b} = c + \frac{d}{b} = c + \left(e + \frac{f}{b} \right)$$

Brent & Kung's algorithm for non-linear equations

1. reduce $y''(t) + a(t)y'(t) + b(t)y(t) = 0$ to a 1st-order equation

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

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

2. reduce the 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:

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

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

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

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 .

First 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]

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

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

Second application: solving 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 $U' - AU = Y'_\kappa - AY_\kappa \pmod{t^{2^{\kappa+1}}}$
- using Lagrange's method of variation of parameters, solution $U = Y_\kappa V_\kappa \pmod{t^{2^{\kappa+1}}}$, $Y'_\kappa - AY_\kappa = Y_\kappa V'_\kappa \pmod{t^{2^{\kappa+1}}}$.

This yields the BCOSSS iteration for Y :

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

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

Further questions

constant factor improvements: middle products of polynomial matrices

small characteristic case: Padé approximants? p -adic lifting?

faster Newton for the case of a single equation: exploit companion form

bit complexity analysis