Fast computation of power series solutions of systems of differential equations

B. Salvy, F. Chyzak, F. Ollivier, B. Salvy, A. Sedoglavic

Joint work with

A. Bostan, E. Schost, A. Sedoglavic

ALGO, INRIA Rocquencourt
fast means using few operations (+, ×, ÷) in the base field $K$.
Given a linear differential equation with power series coefficients:

\[ \mathbf{A} \mathbf{y} = \mathbf{0} \]

and for non-linear systems:

\[ \mathcal{N} \mathbf{y} = \mathbf{0} \]

More precisely:

- Compute the first \( N \) terms of a (basis of) power series solution(s):
  \[ (t) \hat{y}(t) \]

- Naive algorithm (undetermined coefficients):
  \( O(2^N r) \)

- Best that can be hoped: complexity linear in \( N \) and polynomial in \( r \).

Same problem for linear systems:

\[ \mathbf{Y}' = \mathbf{A} \mathbf{Y} \]

and for non-linear systems.
Fast polynomial (matrix) multiplication

- Complexity of polynomial multiplication in degree $\leq N$ by the naive algorithm $= O(N^2)$
- Complexity by the Karatsuba algorithm $= O(N^{\log_2(2\alpha-1)})$
- Complexity by the Toom-Cook algorithm $= O(N^{1.58})$
- Complexity by the Cantor-Kaltofen algorithm $= O(N^\omega M(N))$
- Complexity by the B.-Schost algorithm $= O(N^\omega M(N) \log \log N)$
- Complexity by the Schönhage–Strassen FFT

Fast polynomial (matrix) multiplication
Relaxed multiplication

\[
\mathcal{O}(N \log(N)^{1+\varepsilon} N^{1+\varepsilon}) \quad \text{solution in } O(2002)
\]

\[
\mathcal{O}(N^{1+\varepsilon} N^{1+\varepsilon}) \quad \text{solution in } O(2002)
\]

Previous results

\[
\mathcal{O}(N \log N) \quad \text{van der Hoeven (2002)}
\]

\[
\mathcal{O}(N^{1+\varepsilon}) \quad \text{van der Hoeven (2002)}
\]

\[
\mathcal{O}(N \log N) \quad \text{van der Hoeven (2002)}
\]

\[
\mathcal{O}(N^{1+\varepsilon}) \quad \text{van der Hoeven (2002)}
\]

\[
\mathcal{O}(N^{1+\varepsilon} N^{1+\varepsilon}) \quad \text{solution in } O(2002)
\]

\[
\mathcal{O}(N^{1+\varepsilon} N^{1+\varepsilon}) \quad \text{solution in } O(2002)
\]
New results

<table>
<thead>
<tr>
<th>( (N \cdot t) \mathcal{O} )</th>
<th>( (N, r) \mathcal{O} )</th>
<th>( (N, t, p) \mathcal{O} )</th>
<th>( (N, t) \mathcal{O} )</th>
<th>( (N, t) \mathcal{O} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( (N, t) \mathcal{O} )</td>
<td>( (N, t, p) \mathcal{O} )</td>
<td>( (N, t, p) \mathcal{O} )</td>
<td>( (N, t) \mathcal{O} )</td>
<td>( (N, t) \mathcal{O} )</td>
</tr>
<tr>
<td>( (N, t) \mathcal{O} )</td>
<td>( (N, t, p) \mathcal{O} )</td>
<td>( (N, t) \mathcal{O} )</td>
<td>( (N, t) \mathcal{O} )</td>
<td>( (N, t) \mathcal{O} )</td>
</tr>
</tbody>
</table>

- \( (N, t) \mathcal{O} \): \(O(M^r N/r)\)
- \( (N, t, p) \mathcal{O} \): \(O(d^2 N)\)
- \( (N, t, p) \mathcal{O} \): \(O(M^r N \log N)\)
- \( (N, t) \mathcal{O} \): \(O(r^2 N)\)
- \( (N, t) \mathcal{O} \): \(O(N)\)

- \( (N, r) \mathcal{O} \): \(O(M^r)\)
- \( (N, r) \mathcal{O} \): \(O(d^2 N)\)
- \( (N, r) \mathcal{O} \): \(O(r^2 M(N) \log N)\)
- \( (N, r) \mathcal{O} \): \(O(r N)\)
- \( (N, r) \mathcal{O} \): \(O(N)\)

- \( (N, t, p) \mathcal{O} \): \(O(d N)\)
- \( (N, t, p) \mathcal{O} \): \(O(d N)\)
- \( (N, t, p) \mathcal{O} \): \(O(d N)\)
- \( (N, t, p) \mathcal{O} \): \(O(d N)\)
- \( (N, t, p) \mathcal{O} \): \(O(d N)\)

- \( (N, t) \mathcal{O} \): \(O(d N)\)
- \( (N, t) \mathcal{O} \): \(O(d N)\)
- \( (N, t) \mathcal{O} \): \(O(d N)\)
- \( (N, t) \mathcal{O} \): \(O(d N)\)
- \( (N, t) \mathcal{O} \): \(O(d N)\)

- \( (N, t) \mathcal{O} \): \(O(d N)\)
- \( (N, t) \mathcal{O} \): \(O(d N)\)
- \( (N, t) \mathcal{O} \): \(O(d N)\)
- \( (N, t) \mathcal{O} \): \(O(d N)\)
- \( (N, t) \mathcal{O} \): \(O(d N)\)
Experimental results

---

---
### Experimental Results

<table>
<thead>
<tr>
<th>Basis, system with &amp; equations, precision N: new vs. naive</th>
<th>92 vs. 1/2 day</th>
<th>36 vs. &gt; 3h</th>
<th>15 vs. 2795</th>
<th>6.41 vs. 688</th>
<th>3 vs. 169</th>
<th>512 vs. 688</th>
<th>1024 vs. 2795</th>
</tr>
</thead>
<tbody>
<tr>
<td>92 vs. 1/2 day</td>
<td>13.69 vs. 8348</td>
<td>5.54 vs. 2025</td>
<td>2.30 vs. 484</td>
<td>1 vs. 113</td>
<td>0.44 vs. 28</td>
<td>0.42 vs. 503</td>
<td>0.42 vs. 503</td>
</tr>
<tr>
<td>36 vs. &gt; 3h</td>
<td>2.26 vs. 1686</td>
<td>0.94 vs. 424</td>
<td>0.39 vs. 104</td>
<td>0.17 vs. 26</td>
<td>0.08 vs. 6</td>
<td>0.18 vs. 32</td>
<td>0.18 vs. 32</td>
</tr>
<tr>
<td>15 vs. 2795</td>
<td>1.13 vs. 2795</td>
<td>0.39 vs. 104</td>
<td>0.39 vs. 104</td>
<td>0.17 vs. 25</td>
<td>0.08 vs. 6</td>
<td>0.08 vs. 32</td>
<td>0.08 vs. 32</td>
</tr>
<tr>
<td>6.41 vs. 688</td>
<td>0.44 vs. 28</td>
<td>6.41 vs. 2025</td>
<td>5.54 vs. 2025</td>
<td>2.30 vs. 484</td>
<td>1 vs. 113</td>
<td>0.42 vs. 503</td>
<td>0.42 vs. 503</td>
</tr>
<tr>
<td>3 vs. 169</td>
<td>512 vs. 688</td>
<td>1024 vs. 2795</td>
<td>1024 vs. 2795</td>
<td>6.41 vs. 688</td>
<td>3 vs. 169</td>
<td>13.69 vs. 8348</td>
<td>13.69 vs. 8348</td>
</tr>
</tbody>
</table>

N: **Experiment**
Experimental results

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

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

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

DAC.dat

naive.dat
The divide-and-conquer algorithm

Problem: solve $L y = 0$, where $L = \sum_i a_i (t) \delta_i$, 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 $L(\delta) y = L(\delta) y_0 + t m L(\delta + m) y_1$.

DAC algorithm to solve $L(\delta) y = 0 \mod t^2 m$:

- determine $y_0$, by recursively solving $L(\delta) y_0 = 0 \mod t m$;
- compute the "error" $R$, such that $L(\delta) y_0 = t m R \mod t^2 m$;
- compute $y_1$, by recursively solving $L(\delta + m) y_1 = -R \mod t m$.

$C(N) = 2 C(N/2) + O(r M(N)) \bar{C}(N) = O(r M(N) \log N)$

$\triangleright$ 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:

$\tilde{C}(N) = \tilde{C}(N/2) + O(M(r, N)) \bar{\tilde{C}}(N) = O(M(r, N))$
The divide-and-conquer algorithm

Problem: solve $L y = 0$, where $L = \sum_{i} a_i (t) \delta_i$, 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 $L(y_0) = L(y_1) = 0$.

DAC algorithm to solve $L(y_0) = 0$:

1. Determine $y_0$ by recursively solving $L(y_0) = 0$.
2. Compute the "error" $R$, such that $L(y_0) = t m R$.
3. Compute $y_1$ by recursively solving $L(y_1) = -R$.

$C(N) = 2 C(N/2) + O(r M(N))$. 
$\bar{C}(N) = O(r M(N) \log N)$.

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:

$\tilde{C}(N) = \tilde{C}(N/2) + O(M(r, N))$.

Proof: if $h_t y_1 = h_t w t = 0$, then $h_t y_1 + h_t w t = h_t y_0$. Then $y_0 = h_t y_1$.

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

Problem: solve $\frac{wp}{p} \cdot t = 0$, where $y = h y_1$.

The divide-and-conquer algorithm
The divide-and-conquer algorithm solves the problem of finding the roots of a polynomial

\[ L_y = 0 \]

where

\[ L = \sum a_i(t) \delta_i \]

with \( \delta = \frac{d}{dt} \).

**Idea:** The lowest terms of \( y(t) \) only depend on the lowest terms of \( a_i \).

**Proof:** If \( y = y_0 + tm \), then

\[ L(\delta) y = L(\delta) y_0 + tm \]

DAC algorithm to solve \( L(\delta) y = 0 \mod t^2m \):

1. determine \( y_0 \), by recursively solving \( L(\delta) y_0 = 0 \mod t^m \);
2. compute the "error" \( R \), such that \( L(\delta + m) y = R \mod t^m \);
3. compute \( y_1 \), by recursively solving \( L(\delta + m) y = -R \mod t^m \).

**Newton method:** Computing a whole basis of solutions in 1. will allow to determine \( y_1 \) from \( y_0 \) and \( R \) alone, without a second recursive call:

\[ (N \log(N)) C = 2(N/2) C + O(rM(N)) \]

\[ \bar{C}(N) = O(rM(N) \log N) \]

Problems: Solve: solve \( \frac{dp}{dt} f = 0 \), where \( f(\tau)^{2\sqrt{\tau}} = \tau \), with

\[ y = \psi(\tau) x \]
Newton iteration: real case

\[ x_{\kappa + 1} = x_{\kappa} - \frac{(x_{\kappa}^2 \kappa - 2)}{2x_{\kappa}}, \quad x_0 = 1.5 \]

\[ x_1 = 1.4166666666666666666666666667 \]
\[ x_2 = 1.41421356237445098039215686274510 \]
\[ x_3 = 1.41421356237445098039215686274510 \]
\[ x_4 = 1.41421356237445098039215686274510 \]

\[ 1 + x = 1.5 \]

Newton iteration: real case
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:

\[
[\hat{t}] \mathbb{K}[[t]] \text{ or solve } \hat{g} = \frac{(\hat{g})' \hat{t}}{(\hat{g})}.
\]

Let \( \varphi: \mathbb{K}[[t]] \rightarrow \mathbb{K}[[t]] \) be an \( \mathbb{K}[[t]] \)-valued \( \varphi \) such that \( \varphi(0) = 0 \).

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 \( O(N) \) operations in \( \mathbb{K} \).

The number of coefficients doubles after each iteration.

Total cost = \( 2 \times \) the cost of the last iteration.

\[
\text{mod } \frac{(\hat{t}^N)'}{(\hat{t}^N)} - \frac{(t^N)}{(t^N)} = \hat{t}^{N+1}
\]

Newton iteration: power series case
Division and logarithm of power series

1. Computing the Taylor expansion of \( \frac{f}{f} \) modulo \( t^{N-1} \):

\[
\phi \left( (N)(\mathbb{W}) \mathcal{O} \right) \in [[t]] \mathbb{K} t + 1 \in \mathcal{O} \text{ of } f - 1 \frac{2}{t} \frac{1}{t^2} \frac{1}{t^2} - = (f) \log
\]

2. Taking the antiderivative of \( h \):

\[
\text{(division of power series at precision in } N)\]

0. For \( 0 \leq \gamma < N \) mod \( t^{1+2\gamma} \), \( \lambda f - \lambda \mathcal{O} = 1+\lambda \mathcal{O} \) and \( \frac{0f}{1} = 0 \mathcal{O} \)

\[
f - \delta / 1 = (\delta) \phi \text{ choose } f \in [[t]] \mathbb{K} \in \mathcal{O} \text{ of } f \text{ to compute the reciprocal of power series}
\]
Exponentials of power series

◮ To compute the first $N$ terms of the exponential $\exp(f) = \sum_{i \geq 0} 1/i! f^i$

◮ Choose $\varphi(g) = \log(g) - f$. Iteration: $g_0 = 1$ and $g_{\kappa+1} = g_\kappa - g_\kappa (\log(g_\kappa) - f) \mod t^{\kappa+1}$ for $\kappa \geq 0$.

◮ First order differential equations: compute the first $N$ terms of $f \in \mathbb{K}[[t]]$ such that $af' + bf = c$.

• if $c = 0$ then the solution is $f = \exp(-\int b/a)$. • else, variation of constants: $f = f_0 g$, where $g' = c/a f_0$.

First order differential equations: compute the first $N$ terms of $f \in \mathbb{K}[[t]]$ such that $af' + bf = c$.

• if $c = 0$ then the solution is $f = \exp(-\int b/a)$. • else, variation of constants: $f = f_0 g$, where $g' = c/af_0$.

◮ To compute the first $N$ terms of the exponential $\exp(f)$, for $\kappa \geq 0$.

Exponentials of power series
Intermezzo: constant coefficients case

Problem: solve

\[ y'(t) = Ay(t), \quad y(0) = v, \]

where \( A \) is a scalar matrix.

Equivalent question: compute

\[ y(t) = \exp(\int A v) . \]

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

Idea: the Laplace transform

\[ z_N = \sum_{i=0}^{N-1} A_i v^i. \]

Algorithm (sketch):

1. Compute \( z \) as a rational function of degree \( \leq r \) (indep. on \( N \));
2. Deduce its Taylor expansion modulo \( t^N \) to expand each coordinate of \( z \).

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

Equivalemt question: compute \( \exp(\int A v) \).

Problem: solve \( y = (0) v, \) where \( A \) is a scalar matrix.
Brent & Kung's algorithm for non-linear equations

1. \[ ((N)'I - \mu)\text{Lin} \Rightarrow (N, I - \mu)\text{Lin} = (N, I - \mu)\text{Lin} + (N)\text{Lin} + (N)'I \text{Lin} \]

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:

\[
0 = q - \zeta L - LP + QL \quad \text{thus} \quad q = LS + \zeta L'P = L + S
\]

\[
((t)L + A)((t)S + A) \quad \text{as} \quad (t)q + A(t)p + \zeta A \text{ factor}
\]

\[
(t)f(t)q + (t)f(t)p + (t)'f(t) = 0 \quad \text{to a 1st order equation}
\]

Brent & Kung's algorithm for non-linear equations
Then, the sequence $ \mathcal{X} \text{ converges quadratically to the solution } \mathcal{X}.$

\[
(\mathcal{X})\phi = \nabla \cdot \mathcal{X}|\phi \delta \\
(\mathcal{X})\phi = \nabla \cdot \mathcal{X}|\phi \delta
\]

is a solution of valuation $\geq 2_{t+1}$ of the linearized equation.

Define the sequence $\mathcal{X}^{t+1} - \mathcal{X}^{t} \ni \mathcal{W}^{t+1} \ni \mathcal{W}^{t}$, where

Suppose we have to solve a "functional" equation of the form $f(\mathcal{X}) = 0$, where

Newton iterates hit again.
First application: matrix inversion

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

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

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

$c_{\text{inv}}(N) = c_{\text{inv}}(N/2) + O(M(r, N))$
$ar{c}_{\text{inv}}(N) = O(M(r, N))$

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

First application: matrix inversion
Second application: solving differential equations

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

- Choose the map $\phi$: $Y \rightarrow Y' - AY$.

- The equation for $U$ is $U' - AU = Y'$\(\mod (I^2 + 1)\)\(\kappa\).

- Using Lagrange's method of variation of parameters, solution $U = Y\kappa V\kappa \mod (I^2 + 1)$

This yields the BCOSS iteration for $Y$:

- $Y_{\kappa + 1} = Y_{\kappa} \int Y'_{\kappa} - Y_{\kappa} A_{\kappa} \mod (I^2 + 1)$

solving differential equations

- Choose the map $\phi$ of the system $Y$, \(\kappa\)

$Y' = AY$

$N$ solve $\left(1 + \frac{1}{N}\right)$

$C_{\text{solve}}(N) = C_{\text{solve}}(N/2) + O(M(r, N))$

$\bar{C}_{\text{solve}}(N) = O(M(r, N))$
Further questions

canstant 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