Online Hensel Lifting for Dense, Sparse and Structured Linear System Solving*

by Romain Lebreton

Université Montpellier II

Joint work in progress with
J. Berthomieu, Université Pierre et Marie Curie
É. Schost, University of Western Ontario

SIAM AG'13 – Fort Collins
August 3, 2013

*. This document has been written using the GNU $\TeX_{\text{MACS}}$ text editor (see www.texmacs.org).

Problem statement

(Simplified) Problem.

Given $A \in \mathcal{M}_{r\times 1}(k[x]_{<d})$ and $B \in \mathcal{M}_{r\times r}(k[x]_{<d})$ invertible over $k[[x]]$,

solve the linear system $A = B \cdot C$ for $C \in \mathcal{M}_{r\times 1}(k[[x]])$ at precision $N \geqslant d$.

Application.

- Power series linear system solving $(N = d)$
- Polynomial linear system solving with $C \in \mathcal{M}_{r\times 1}(k(x))$ $(N = 2\,r\,d)$

Outline of the talk

1. **Overview of general linear system solver over $k[[x]]$**
   a. Dixon's algorithm
   b. Moenck-Carter algorithm
   c. Our contribution : Online algorithm
   d. Newton's iteration

2. Application to different matrix representations
   a. Dense matrices
   b. Structured matrices
   c. Sparse matrices

Dixon's algorithm

Notations.

- Let $C = \sum_{i \in \mathbb{N}} C_i \, x^i$

Algorithm - [Dixon, '82]

Input: $A \in \mathcal{M}_{r \times 1}(k[[x]])$, $B \in \mathcal{M}_{r \times r}(k[[x]])$ and $N \in \mathbb{N}$
Output: $C \in \mathcal{M}_{r \times 1}(k[[x]])$ such that $A = B \cdot C \bmod x^N$

1. $\Gamma = B^{-1} \bmod x$
2. $C_0 := (\Gamma \cdot A) \bmod x$
3. **for** i from 1 to $N-1$
   a. $A := (A - B \cdot C_{i-1}) \operatorname{quo} x$ (Update the system)
   b. $C_i := (\Gamma \cdot A) \bmod x$ (Find one term of C)
4. **return** $C := \sum_{i=0}^{N-1} C_i \, x^i$

Dixon's algorithm - Cost

Focus on polynomial arithmetic:

- At step i, we compute among other thing

$$B \cdot C_{i-1} = (B_0 \cdot C_{i-1} + \cdots + B_{d-1} \cdot C_{i-1}\, x^{d-1})$$

- Arrived at precision N, we have computed $\sum_{\substack{0 \leqslant j < d \\ 0 \leqslant k < N}} B_j \cdot C_k \simeq B \cdot C \bmod x^N$

Cons. Naive polynomial arithmetic

Pros. Requires the inverse matrix B^{-1} at precision 1

Moenck-Carter's algorithm

- Let $C_{i\ldots j} = C_i + C_{i+1}\,x + \cdots + C_{j-1}\,x^{j-i-1}$
- Moenck-Carter's algorithm is an x^d-adic version of Dixon's algorithm.

Algorithm - [Moenck, Carter, '79]

Input: $A \in \mathcal{M}_{r\times 1}(k[[x]])$, $B \in \mathcal{M}_{r\times r}(k[[x]])$ and $N \in \mathbb{N}$
Output: $C \in \mathcal{M}_{r\times 1}(k[[x]])$ such that $A = B \cdot C \bmod x^N$

1. $\Gamma = B^{-1} \bmod x^d$
2. $C_{0\ldots d} := (\Gamma \cdot A) \bmod x^d$

3. **for** i from 1 to $N/d - 1$
   a. $A := (A - B \cdot C_{(i-1)d\ldots id}) \operatorname{quo} x^d$ (Update the system)
   b. $C_{id\ldots(i+1)d} := (\Gamma \cdot A) \bmod x^d$ (Find d terms of C)

4. **return** $C := \sum_{i=0}^{N/d-1} C_{id\ldots(i+1)d}\,(x^d)^i$

Moenck-Carter's algorithm – Cost

Focus on polynomial arithmetic:

- At step i, we compute among other thing $B_{0\ldots d} \cdot C_{(i-1)d\ldots id}$
- Arrived at precision N, we have computed

$$\sum_{0 \leqslant k < N/d} B_{0\ldots d} \cdot C_{(i-1)d\ldots id} \simeq B \cdot C \bmod x^N$$

Pros. Fast polynomial arithmetic

Cons. Requires the inverse matrix B^{-1} at precision d

Dixon's algorithm revisited.

Compute iteratively the power series coefficients C_i using Formula (?)

$$\begin{aligned} B \cdot C &= A \\ \Leftrightarrow \quad B_0 \cdot C &= A - (B - B_0) \cdot C \\ \Leftrightarrow \quad C &= B_0^{-1} \cdot \left(A - x \left(\left(\frac{B - B_0}{x} \right) \cdot C \right) \right) \end{aligned} \tag{1}$$

i.e.

1. $C_1 = B_0^{-1} \cdot \left(A_1 - \left(\left(\frac{B - B_0}{x} \right) \cdot C \right)_0 \right) = B_0^{-1} \cdot (A_1 - B_1 C_0)$,
2. $C_2 = B_0^{-1} \cdot \left(A_2 - \left(\left(\frac{B - B_0}{x} \right) \cdot C \right)_1 \right) = B_0^{-1} \cdot (A_2 - (B_2 C_0 + B_1 C_1))$,
3. $C_3 = B_0^{-1} \cdot \left(A_3 - \left(\left(\frac{B - B_0}{x} \right) \cdot C \right)_2 \right) = B_0^{-1} \cdot (A_3 - (B_3 C_0 + B_2 C_1 + B_1 C_2))$,
4. ...

Focus on the problematic product

In Dixon's, the matrix-vector product $D = \left(\left(\frac{B - B_0}{x}\right) \cdot C\right)$ over $k[[x]]$ must satisfy:

1. coefficients D_i must be computed iteratively, one coefficient at a time
2. when computing D_i, we should only require $C_0, \ldots, C_i$

Focus on the problematic product

In Dixon's, the matrix-vector product $D = \left(\left(\frac{B - B_0}{x} \right) \cdot C \right)$ over $k[[x]]$ must satisfy:

1. coefficients D_i must be computed iteratively, one coefficient at a time
2. when computing D_i, we should only require $C_0, \ldots, C_i$

This is an online algorithm

Online multiplication

Theorem. [Fischer, Stockmeyer '74], [Schröder '97], [Hoeven '97], [Berthomieu, Hoeven, Lecerf '11], [L., Schost '13]

Multiply $B \cdot C$ over $R[[x]]$ at precision N by an online algorithm takes

$$\mathsf{R}(N) = \tilde{\mathcal{O}}(N).$$

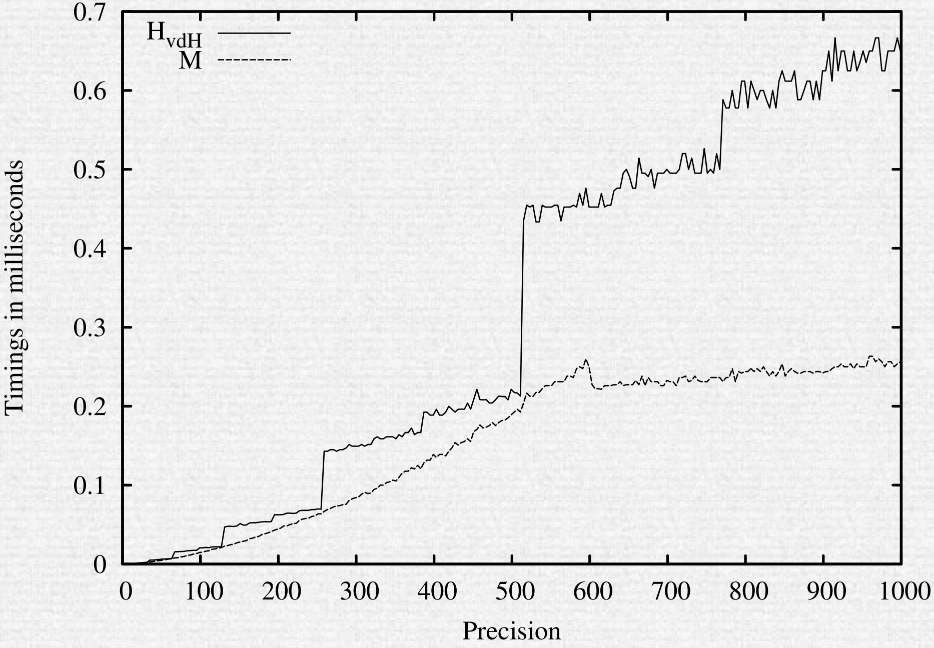

Figure. Timings of [Hoeven '03] online multiplication in $k[[x]]$

Online solver for linear system

Online linear solver. [Berthomieu, L. '12], [Berthomieu, L., Schost '13]

Compute the coefficients C_i iteratively using

$$C = B_0^{-1} \cdot \left(A - x \left(\left(\frac{B - B_0}{x} \right) \cdot C \right) \right)$$

and a fast online multiplication for $\left(\frac{B - B_0}{x} \right) \cdot C$.

Pros. Fast online polynomial arithmetic

Pros. Requires the invert matrix B^{-1} at precision 1

Newton iteration

Algorithm – Newton
1. Compute $(B^{-1})_{0\dots N/2}$ using Newton iteration 2. **for** i from 1 to $\lceil \log_2(N) \rceil$ i. $C_{0\dots 2^i} = C_{0\dots 2^{i-1}} - (B^{-1})_{0\dots 2^{i-1}} \cdot (B_{0\dots 2^i} \cdot C_{0\dots 2^{i-1}} - A_{0\dots 2^i}) \bmod x^{2^i}$ 3. **return** C

Pros. Fast polynomial arithmetic

Cons. Requires the inverse matrix B^{-1} at precision $N/2$

Outline of the talk

1. Overview of general linear system solver over $k[[x]]$
    a. Dixon's algorithm
    b. Moenck-Carter algorithm
    c. Our contribution : Online algorithm
    d. Newton's iteration

2. **Application to different matrix representations**
    a. Dense matrices
    b. Structured matrices
    c. Sparse matrices

Dense matrices

Algorithm	$N = d$	$N = r\,d$
[Dixon '82]	$\tilde{\mathcal{O}}(r^{\omega} + r^2\, \underline{d}^2)$	$\tilde{\mathcal{O}}(r^3\, d^2)$
[Moenck, Carter '79]	$\tilde{\mathcal{O}}(r^{\omega}\, \underline{d})$	$\tilde{\mathcal{O}}(r^3\, d)$
Newton iteration	$\tilde{\mathcal{O}}(r^{\omega}\, \underline{d})$	$\tilde{\mathcal{O}}(r^{\omega+1}\, d)$
Our online algorithm	$\tilde{\mathcal{O}}(r^{\omega} + r^2\, d)$	$\tilde{\mathcal{O}}(\underline{r}^3\, d)$
[Storjohann '03]	$\tilde{\mathcal{O}}(r^{\omega}\, \underline{d})$	$\tilde{\mathcal{O}}(r^{\omega}\, d)$

Table. Costs of solving $A = B \cdot C$ for dense matrices over $\mathbb{k}[[X]]$

Timings –

- p-adic integers linear system solving ($N = d$) with $p = 536871001$
- Timings in milliseconds
- Online algorithms implemented inside MATHEMAGIX

N	4	16	64	256	1024	4096
LINBOX	1.4	3.6	25	310	4700	77000
Online	0.24	0.58	2.1	14	110	760

Table. Integer linear system of size $r = 4$

N	4	16	64	2256	1024
LINBOX	25	170	1900	27000	480000
Online	150	360	2000	14000	90000

Table. Integer linear system of size $r = 32$

Notation: Let $L(\boldsymbol{v}) = \begin{pmatrix} v_0 & 0 & \cdots & 0 \\ v_1 & v_0 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ v_{r-1} & \cdots & v_1 & v_0 \end{pmatrix}$, $U(\boldsymbol{v}) = {}^t L(\boldsymbol{v})$ where $\boldsymbol{v} = (v_0, \ldots, v_{r-1})$.

Definition.

The matrix $B \in \mathcal{M}_{r\times r}(k[[x]])$ is *structured* of displacement rank α if

$$\exists \boldsymbol{v}_i, \boldsymbol{w}_i \in k[[x]]^r \text{ s.t. } B = \sum_{i=1}^{\alpha} L(\boldsymbol{v}_i)\, U(\boldsymbol{w}_i).$$

Here we consider quasi-Toeplitz matrices *i.e.* “close” to $\begin{pmatrix} b_r & \ldots & b_{2r-2} & b_{2r-1} \\ \vdots & b_r & & b_{2r-2} \\ b_2 & & \ddots & \vdots \\ b_1 & b_2 & \ldots & b_r \end{pmatrix}$.

Notation: Let $L(\boldsymbol{v}) = \begin{pmatrix} v_0 & 0 & \cdots & 0 \\ v_1 & v_0 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ v_{r-1} & \cdots & v_1 & v_0 \end{pmatrix}$, $U(\boldsymbol{v}) = {}^t L(\boldsymbol{v})$ where $\boldsymbol{v} = (v_0, \ldots, v_{r-1})$.

Definition.

The matrix $B \in \mathcal{M}_{r\times r}(k[[x]])$ is *structured* of displacement rank α if

$$\exists \boldsymbol{v}_i, \boldsymbol{w}_i \in k[[x]]^r \text{ s.t. } B = \sum_{i=1}^{\alpha} L(\boldsymbol{v}_i)\, U(\boldsymbol{w}_i).$$

Cost of basic matrix operations.

- Matrix-vector product $B \cdot C$ — $\tilde{\mathcal{O}}(\alpha\, r)$
- Cost of precomputation of B^{-1} — $\tilde{\mathcal{O}}(\alpha^2\, r)$
- Cost of matrix-vector product $B^{-1} \cdot C$ — $\tilde{\mathcal{O}}(\alpha\, r)$

Structured matrices

Let d' such that $x_i, y_i \in k[x]_{<d'}$ in the decomposition $B = \sum_{i=1}^{\alpha} L(x_i)\, U(y_i)$.

Algorithm	$N = d'$	$N = r\, d'$
[Dixon '82]	$\tilde{\mathcal{O}}(\alpha^2 r + \alpha\, r\, \underline{d'^2})$	$\tilde{\mathcal{O}}(\alpha\, r^2\, \underline{d'^2})$
[Moenck, Carter '79]	$\tilde{\mathcal{O}}(\underline{\alpha^2}\, r\, d')$	$\tilde{\mathcal{O}}(\alpha\, r^2\, d')$
Newton iteration	$\tilde{\mathcal{O}}(\underline{\alpha^2}\, r\, d')$	$\tilde{\mathcal{O}}(\underline{\alpha^2}\, r^2\, d')$
Our online algorithm	$\tilde{\mathcal{O}}(\alpha^2 r + \alpha\, r\, d')$	$\tilde{\mathcal{O}}(\alpha\, r^2\, d')$

Table. Costs of solving $A = B \cdot C$ for structured matrices over $\mathbb{k}[[X]]$

Question: Is $\left(\frac{B - B_0}{x}\right)$ structured ?

Sparse matrices

Let B be a sparse matrix with $\tilde{\mathcal{O}}(r)$ non-zero elements

State of the art.

1. [Wiedemann '86] + [Dixon, '82]

   Wiedemann's algorithm to solve $A = B \cdot C$ over k.

   Cost of $B^{-1}A$ over k: $\mathcal{O}(r^2)$ (+ Precomputation)

2. [Eberly, Giesbrecht, Giorgi, Storjohann, Villard '07] + [Dixon, '82]

   Cost of $B^{-1}A$ over k: $\tilde{\mathcal{O}}(r^{1.5})$ (+ Precomputation)

Our candidate algorithm.

[Eberly, Giesbrecht, Giorgi, Storjohann, Villard '07] + Online linear solver

Conclusion and Perspectives

Conclusion.

- Online algorithm brings the best of Dixon's and Moenck-Carter algorithms

Work in progress.

- Sparse / blackbox matrices
- Implementation

Perspectives.

- Hybrid Newton / online algorithm
- High-order lifting in online algorithms

Thank you

for your attention

Question: Is $\left(\frac{B-B_0}{x}\right)$ structured ?

YES since $$L(v)\,U(w) = \underbrace{L(v_0)\,U(w_0)}_{B_0} + x\left(\underbrace{L(v)\,U\Big(\frac{w-w_0}{x}\Big) + L\Big(\frac{v-v_0}{x}\Big)\,U(w_0)}_{(B-B_0)/x}\right).$$