# Algorithmes détendus rapides pour la remontée de Hensel $p$-adique et applications aux systèmes algébriques*

Jérémy Berthomieu

Laboratoire de Mathématiques
Université de Versailles – St-Quentin-en-Yvelines

Romain Lebreton

Laboratoire d'Informatique
École polytechnique

Séminaire Algorithms – Rocquencourt

Lundi 13 février 2012

*. This document has been written using the GNU TeXmacs text editor (see `www.texmacs.org`).

# Lazy algorithms in $\mathbb{Z}_p$

**Definition 1.** *A $p$-adic integer is an element written*

$$\sum_{n=0}^{\infty} a_n\, p^n, \quad a_n \in \{0, \ldots, p-1\}.$$

*A lazy $p$-adic integer is an algorithm for $n \longmapsto a_n$.*

**Example 2. (of a session of Mathemagix)**

```
Mmx] use "algebramix"; p_adic (a, p) == p_adic (@p_expansion (a, modulus p));
     x == p_adic (532, 10)
```

$2 + 3\,p + 5\,p^2 + O(p^{10})$

```
Mmx] x == -p_adic (1, 10)
```

$9 + 9\,p + 9\,p^2 + 9\,p^3 + 9\,p^4 + 9\,p^5 + 9\,p^6 + 9\,p^7 + 9\,p^8 + 9\,p^9 + O(p^{10})$

```
Mmx] x == p_adic (1, 13) / p_adic (9876543210^1000, 13)
```

$1 + p + p^2 + 4\,p^3 + 7\,p^4 + p^5 + 12\,p^6 + 9\,p^7 + 4\,p^8 + 11\,p^9 + O(p^{10})$

```
Mmx] [x[10], x[20], x[50], x[100], x[200], x[500], x[1000], x[2000], x[5000]]
```

$[12, 0, 11, 4, 4, 11, 0, 7, 2]$

# Lazy addition in $\mathbb{Z}_p$

```
class Sum_Padic_rep_p ⩾ Padic_rep_p
    a, b: Padic_p
    γ: ℤ
    constructor (ã: Padic_p, b̃: Padic_p)
        a := ã; b := b̃; γ := 0
    method next()
        t := a_n + b_n + γ
        γ := quo(t, p)
        return rem(t, p)
```

To speed up the computation, the next() method can be rewritten

```
    method next()
        t := a_n + b_n + γ
        if t < p then
            γ := 0
            return t
        else
            γ := 1
            return t − p
```

# Motivation

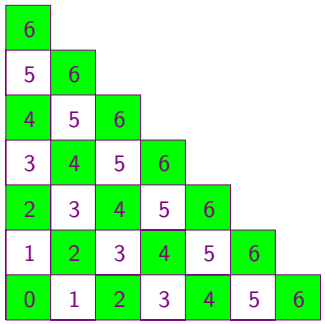


**Figure 1.** Naive multiplication

step $0$: $a_0\,b_0$
step $1$: $a_0\,b_1 + a_1\,b_0$
step $2$: $a_0\,b_2 + a_1\,b_1 + a_2\,b_0$
step $3$: $a_0\,b_3 + a_1\,b_2 + a_2\,b_1 + a_3\,b_0$
step $4$: $a_0\,b_4 + a_1\,b_3 + a_2\,b_2 + a_3\,b_1 + a_4\,b_0$
step $5$: $a_0\,b_5 + a_1\,b_2 + a_2\,b_3 + a_3\,b_4 + a_5\,b_0$
step $6$: $a_0\,b_6 + a_1\,b_5 + a_2\,b_4 + a_3\,b_3 + a_4\,b_2 + a_5\,b_1 + a_6\,b_0$

**Definition 3.** *Two integers in $\mathbb{Z}$ of bit-size less than $m$ can be multiplied in $\mathsf{I}(m) \in O(m \log m\, 2^{\log^* m})$ bit-operations.*

$\rightsquigarrow$ Lazy $p$-adic integers can be multiplied up to precision $n$ in $O(n^2\,\mathsf{I}(\log p))$ bit-operations.

$\rightsquigarrow$ Zealous $p$-adic integers can be multiplied up to precision $n$ in $O(\mathsf{I}(n \log p))$ bit-operations.

# Relaxed multiplications

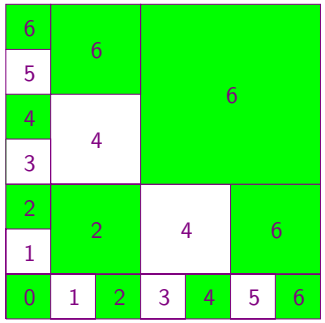


**Figure 2.** Relaxed multiplication

step $0$: $a_0\,b_0$
step $1$: $a_0\,b_1 + a_1\,b_0$
step $2$: $a_0\,b_2 + a_2\,b_0 + (a_1 + a_2\,p)\,(b_1 + b_2\,p)$
step $3$: $a_0\,b_3 + a_3\,b_0$
step $4$: $a_0\,b_3 + a_3\,b_0$
$+ (a_1 + a_2\,p)\,(b_3 + b_4\,p) + (a_3 + a_4\,p)\,(b_1 + b_2\,p)$
step $5$: $a_0\,b_5 + a_5\,b_0$
step $6$: $a_0\,b_6 + a_6\,b_0 + (a_1 + a_2\,p)\,(b_5 + b_6\,p)$
$+ (a_5 + a_6\,p)\,(b_1 + b_2\,p)$
$+ (a_3 + \cdots + a_6\,p^3)\,(b_3 + \cdots + b_6\,p^3)$

**Proposition 4.** [Fischer, Stockmeyer 1974], [van der Hoeven 1997], [Berthomieu, van der Hoeven, Lecerf 2011]
*Let $a$ and $b$ be two relaxed $p$-adic integers known up to precision $n$, then $(a\,b)_0, \ldots, (a\,b)_{n-1}$ can be computed in $\mathsf{R}(n) = O(\mathsf{I}(n \log p) \log n)$ bit-operations.*

# Block representation

A $p$-adic integer $a = \sum_{n=0}^{\infty} a_n p^n$ can be seen as a $p^k$-adic $A = \sum_{n=0}^{\infty} A_n p^{kn}$ by letting

$$A_n = \sum_{i=0}^{k-1} a_{kn+i} p^i.$$

The product of two $p^k$-adic integers can be computed in precision $\frac{n}{k}$ in

- $O\big(\big(\frac{n}{k}\big)^2 \mathsf{I}(\log p^k)\big) \in O\Big(\frac{n^2}{k} \mathsf{I}(\log p)\Big)$ bit-operations using the naive algorithm;
- $O\big(\mathsf{I}\big(\frac{n}{k} \log p^k\big)\big) = O(\mathsf{I}(n \log p))$ bit-operations using the zealous algorithm;
- $O\big(\mathsf{I}\big(\frac{n}{k} \log p^k\big) \log \frac{n}{k}\big) = O\big(\mathsf{I}(n \log p) \log \frac{n}{k}\big) \in \mathsf{R}(n)$ bit-operations using the relaxed algorithm.

## Recursive $p$-adic integer

**Definition 5.** *A $p$-adic integer $a$ is recursive of order $n_0$ if it is solution of an equation $a = \Phi(a)$, where the expression $\Phi(a)_n$ only depends on $a_0, ..., a_{n-1}$ for all $n \geq n_0$.*

**Example 6.** Let $a \in \mathbb{Z}_p$ and $\beta \in \{1, \ldots, p-1\}$. Let us define $\mathsf{mul\_rem}(\beta, a)$ and $\mathsf{mul\_quo}(\beta, a)$ by $(\mathsf{mul\_rem}(\beta, a))_n = \mathsf{rem}(\beta a_n, p)$ and $(\mathsf{mul\_quo}(\beta, a))_n = \mathsf{quo}(\beta a_n, p)$. So that

$$\beta a = \mathsf{mul\_rem}(\beta, a) + p\,\mathsf{mul\_quo}(\beta, a).$$

Let $\gamma = \beta^{-1} \bmod p$. Then, $c = a/\beta$ is recursive of order $1$ and satisfies

$$c = \mathsf{mul\_rem}(\gamma, a - p\,\mathsf{mul\_quo}(\beta, c)), \quad c_0 = a_0\gamma \bmod p.$$

If $a, b \in \mathbb{Z}_p$ are such that $b_0 \neq 0$, then $c = a/b \in \mathbb{Z}_p$ is recursive of order $1$. It is solution of

$$c = \frac{a - (b - b_0)\,c}{b_0}, \quad c_0 = a_0/b_0 \bmod p.$$

$$c = \frac{a - p\left(\left(\frac{b - b_0}{p}\right)c\right)}{b_0}, \quad c_0 = a_0/b_0 \bmod p.$$

Furthermore, $c$ can be computed up to precision $n$ in $\mathsf{R}(n) + O(n)$ bit-operations.

# Timings

| $n$ | 8 | 16 | 32 | 64 | 128 | 256 | 512 | 1024 | 2048 |
|---|---|---|---|---|---|---|---|---|---|
| Naive multiplication | 4 | 7 | 15 | 35 | 95 | 300 | 1000 | 3700 | 14000 |
| Relaxed multiplication | 9 | 21 | 44 | 93 | 200 | 420 | 920 | 2000 | 4800 |
| Relaxed mult. with blocks of size 32 | | | | 65 | 180 | 410 | 900 | 2000 | 4500 |
| Gmp's extended g.c.d. | 3 | 6 | 14 | 35 | 92 | 250 | 730 | 2200 | 5600 |
| Maple 13 | 240 | 320 | 520 | 1200 | 3500 | 11000 | 38000 | 160000 | $\infty$ |
| Pari/Gp | 0.68 | 1.1 | 2.8 | 8.5 | 28 | 99 | 360 | 1300 | 4800 |

**Table 1.** Divisions for $p = 536870923$, in microseconds.

## Recursive $p$-adic integer

**Example 7.** Let $a \in \mathbb{Z}_p$ such that $a_0 \neq 0$ has a $r$th root $b_0$ modulo $p$, then a $r$th root $b$ of $a$ is recursive of order $1$ since

$$b = \frac{a - b^r + r\, b_0^{r-1}\, b}{r\, b_0^{r-1}} \text{ and } b_0^r = a_0 \bmod p.$$

$\rightsquigarrow$ Improvement of formulae found in [van der Hoeven 2002] for $\mathbb{C}[[X]]$.

```
Mmx] minus_one == -p_adic (1, 5)
```

$4 + 4\,p + 4\,p^2 + 4\,p^3 + 4\,p^4 + 4\,p^5 + 4\,p^6 + 4\,p^7 + 4\,p^8 + 4\,p^9 + O(p^{10})$

```
Mmx] i == separable_root (minus_one, 2)
```

$2 + p + 2\,p^2 + p^3 + 3\,p^4 + 4\,p^5 + 2\,p^6 + 3\,p^7 + 3\,p^9 + O(p^{10})$

```
Mmx] soo (n) == set_output_order (x, n); soo (1000); i^2-minus_one
```

$O(p^{1000})$

**Proposition 8.** *Assume that $r$ and $p$ are coprime. If $a \in \mathbb{Z}_p$ and $a_0 \neq 0$ has a $r$th root $b_0$ in $\mathbb{F}_p$, then one can compute the first $n$ terms of $b$ a $r$th root of $a$ in $O(\log r)\,\mathsf{R}(n)$ bit-operations.*

## Shifted algorithms

$\rightsquigarrow$ Whenever $y = \Phi(y)$, we need to explicit the fact that $y_n$ is not needed to compute $(\Phi(y))_n$.

**Example 9.** If $\Phi(Y) = Y^3 + p$, then $\Psi\colon\ Z \mapsto p^3 \left(\frac{Z}{p}\right)^3 + p$ is a *shifted algorithm* for $\Phi$ and $y$, the unique fixed point of $\Phi$ such that $y_0 = 0$.

Another shifted algorithm for $\Phi$ and $y$ is $Z \mapsto p^2 \left(\frac{Z}{p}\right)^2 Z + p$.

**Proposition 10.** *Let $\Phi(Y)$ be a polynomial in $\mathbb{Z}_p[Y]$ and let $y$ be a fixed point of $\Phi$. If $\Psi$ is a shifted algorithm for $\Phi(Y)$ and $y$ with $L^*$ multiplications between $p$-adic integers, then $y$ can be computed at precision $n$ in $L^* \mathsf{R}(n) + O(n)$ bit-operations.*

# Simple root lifting of dense univariate polynomials in $\mathbb{Z}_p$

**Proposition 11.** *Let $P$ be a polynomial in $\mathbb{Z}[Y]$ and let $y_0$ be a simple root of $P$ modulo $p$. Then, there exists a unique $y \in \mathbb{Z}_p$ such that $y = y_0 + O(p)$ and $P(y) = 0$ and it is a fixed-point of*

$$\Phi(Y) = \frac{P'(y_0)\,Y - P(Y)}{P'(y_0)}.$$

*Using the following algorithm, one can compute $y$ at precision $n$ in $(d-1)\,\mathsf{R}(n)$ bit-operations, where $d = \deg P$.*

**Algorithm 1**

**Input.** $P \in \mathbb{Z}[Y]$ with a simple root $y_0$ in $\mathbb{F}_p$.
**Output.** A shifted algorithm $\Psi$ associated to $\Phi$ and $y_0$.

1. Compute $Q(Y)$ the quotient of $P(Y)$ by $(Y - y_0)^2$.
2. Let $\mathrm{sq}(Z)\colon\ Z \mapsto \left(\frac{Z - y_0}{p}\right)^2$.
3. **Return** the shifted algorithm

$$\Psi\colon\ Z \mapsto \frac{-P(y_0) + P'(y_0)\,y_0 - p^2\,(Q(Z)\cdot \mathrm{sq}(Z))}{P'(y_0)}.$$

# Timings

| $n$ | 4 | 16 | 64 | 256 | 1024 | 2048 | 4096 | $2^{14}$ | $2^{16}$ |
|---|---|---|---|---|---|---|---|---|---|
| Naive multiplication | 0.0079 | 0.052 | 0.29 | 2.9 | 39 | 152 | 600 | 9500 | 150000 |
| Naive mult. with blocks of size 32 | | | 0.32 | 0.60 | 2.9 | 8.9 | 31 | 440 | 6700 |
| Naive mult. with blocks of size 1024 | | | | | | 20 | 27 | 120 | 1300 |
| Relaxed multiplication | 0.21 | 0.13 | 0.65 | 2.9 | 14 | 31 | 71 | 400 | 2400 |
| Relaxed mult. with blocks of size 32 | | | 0.33 | 0.73 | 4.1 | 11 | 32 | 240 | 1700 |
| Relaxed mult. with blocks of size 1024 | | | | | | 20 | 30 | 170 | 1400 |
| Newton | 0.0090 | 0.023 | 0.079 | 0.52 | 4.1 | 11 | 29 | 170 | 870 |

**Table 2.** Solving a polynomial of dense size 8 with $p = 536871001$, in milliseconds.

| $n$ | 4 | 16 | 64 | 256 | 1024 | 2048 | 4096 | $2^{14}$ | $2^{16}$ |
|---|---|---|---|---|---|---|---|---|---|
| Naive multiplication | 0.086 | 0.71 | 4.4 | 46 | 640 | 2500 | 9800 | 160000 | $\infty$ |
| Naive mult. with blocks of size 32 | | | 100 | 110 | 140 | 240 | 610 | 8000 | 120000 |
| Naive mult. with blocks of size 1024 | | | | | | 12000 | 13000 | 14000 | 35000 |
| Relaxed multiplication | 0.25 | 2.3 | 12 | 54 | 250 | 560 | 1300 | 7200 | 42000 |
| Relaxed mult. with blocks of size 32 | | | 110 | 110 | 160 | 270 | 600 | 4200 | 30000 |
| Relaxed mult. with blocks of size 1024 | | | | | | 12000 | 13000 | 15000 | 34000 |
| Newton | 0.21 | 0.89 | 8.0 | 86 | 720 | 2000 | 5300 | 30000 | 140000 |

**Table 3.** Solving a polynomial of dense size 128 with $p = 536871001$, in milliseconds.

## Simple root lifting of s.l.p. univariate polynomials in $\mathbb{Z}_p$

For a polynomial $P(Y)$, we define polynomials $T_P(Y) = P(y_0) + P'(y_0)\,(Y - y_0)$ and $E_P(Y) = P(Y) - T_P(Y)$.

**Proposition 12.** *If $P$ is given as a straight-line program (s.l.p.) with $L^*$ multiplications $\cdot$, we define recursively a vector $\tau \in \mathbb{Z}^2$ and a s.l.p. $\varepsilon$. Initially, $\varepsilon^0 := 0$ and $\tau^0 := (y_0, 1)$, then if the $i$th instruction $\Gamma_i$ is*

- $\Gamma_i = (a^c; )$, $\varepsilon^i := 0$, $\tau^i := (a, 0)$;
- $\Gamma_i = (a \times \cdot; u)$, $\varepsilon^i := a\,\varepsilon^u$, $\tau^i := a\,\tau^u$;
- $\Gamma_i = (\pm; u, v)$, $\varepsilon^i = \varepsilon^u \pm \varepsilon^v$, $\tau^i := \tau^u \pm \tau^v$;
- $\Gamma_i = (\cdot; u, v)$, *such that* $\tau^u = (a, c)$ *and* $\tau^v = (b, d)$, *then* $\tau^i := (a\,b, a\,d + b\,c)$ *and*

$$\varepsilon^i := \varepsilon^u \cdot \varepsilon^v + p\left(\left(\frac{c\,\varepsilon^v + d\,\varepsilon^u}{p}\right) \cdot (Z - y_0)\right) + (a\,\varepsilon^v + b\,\varepsilon^u) + p^2\left((c\,d)\left(\frac{Z - y_0}{p}\right)^2\right).$$

*If $P$ has $k$ instructions, then $\varepsilon_P := \varepsilon^k$ is a shifted algorithm for $E_P$ and $y_0$ with $2\,L^* + 1$ multiplications. Moreover, $\tau^k$ is the vector of coefficients of $T_P$ in the basis $(1, Y - y_0)$.*

## Simple root lifting of s.l.p. univariate polynomials in $\mathbb{Z}_p$

**Proposition 13.** *Let $P(Y) \in \mathbb{Z}_p[Y]$ be a univariate polynomial given as a s.l.p. with $L^*$ multiplications between $p$-adic integers. Let $y_0$ be a simple root of $P$ in $\mathbb{F}_p$. Then, the following algorithm:*

$$\Psi\colon\ Z \mapsto \frac{-P(y_0) + P'(y_0)\, y_0 - \varepsilon_P(Z)}{P'(y_0)}$$

*is a shifted algorithm associated to $\Phi$ and $y_0$ whose multiplicative complexity is $2\,L^* + 1$.*

**Theorem 14.** *The root $y \in \mathbb{Z}_p$ of $P$ can be computed at precision $n$ in $(2\,L^* + 1)\,\mathsf{R}(n) + O(n)$ bit-operations.*

# Regular root lifting of multivariate linear systems in $\mathbb{Z}_p$

$\rightsquigarrow$ How to invert a matrix $\mathrm{B} \in \mathcal{M}_r(\{0, ..., p-1\})$ in $\mathcal{M}_r(\mathbb{Z}_p)$?

**Definition 15.** *Operators* Mul_rem *and* Mul_quo *from* $\mathcal{M}_r(\{0, ..., p-1\}) \times \mathcal{M}_{r \times s}(\mathbb{Z}_p)$ *to* $\mathcal{M}_r(\mathbb{Z}_p)$ *are defined by*

$$\begin{aligned} (\mathsf{Mul\_rem}(\mathrm{B}, A))_n &= \mathsf{rem}(\mathrm{B}\, A_n, p); \\ (\mathsf{Mul\_quo}(\mathrm{B}, A))_n &= \mathsf{quo}(\mathrm{B}\, A_n, p). \end{aligned}$$

*So that* $\mathrm{B}\, A = \mathsf{Mul\_rem}(\mathrm{B}, A) + p\, \mathsf{Mul\_quo}(\mathrm{B}, A)$.

**Proposition 16.** *Let us denote* $\mathsf{MM}(r, s)$ *the number of operations in the ground ring to multiply a square matrix of size* $r$ *and a matrix of size* $r \times s$*, then* $\mathsf{MM}(r, s) \in O(r^2\, s^{\omega-2})$ *if* $r \geq s$*, and* $\mathsf{MM}(r, s) \in O(r^{\omega-1}\, s)$ *otherwise.*

*Let* $A \in \mathcal{M}_{r \times s}(\mathbb{Z}_p)$ *and* $\mathrm{B} \in \mathcal{M}_r(\{0, ..., p-1\})$ *invertible modulo* $p$ *with* $\Gamma = \mathrm{B}^{-1} \bmod p$*. Then,* $C = \mathrm{B}^{-1}\, A$ *is recursive of order* $1$ *and satisfies*

$$C = \mathsf{Mul\_rem}(\Gamma, A - p\, \mathsf{Mul\_quo}(\mathrm{B}, C)), \quad C_0 = \Gamma\, A_0 \bmod p.$$

# Regular root lifting of multivariate linear systems in $\mathbb{Z}_p$

**Proposition 17.** *Let $A \in \mathcal{M}_{r\times s}(\mathbb{Z}_p)$ and $B \in \mathcal{M}_r(\mathbb{Z}_p)$ such that $B_0$ is invertible modulo $p$ with $\Gamma = B_0^{-1} \bmod p$. Then, $C = B^{-1}A$ is recursive of order $1$ and satisfies*

$$C = B_0^{-1}\left(A - p\left(\frac{B - B_0}{p}\cdot C\right)\right), \quad C_0 = \Gamma A_0 \bmod p.$$

*Thus, $C$ can be computed up to precision $n$ in $\mathsf{MM}(r,s)\,\mathsf{R}(n) + O(n)$ bit-operations.*

| $n$ | 4 | 16 | 64 | 256 | 1024 | 4096 | $2^{14}$ | $2^{16}$ |
|---|---|---|---|---|---|---|---|---|
| Newton | 0.097 | 0.22 | 0.89 | 6.8 | 59 | 490 | 3400 | 20000 |
| Mmx | 0.15 | 0.61 | 3.1 | 8.1 | 38 | 335 | 1600 | 14000 |
| Variant | Naive | Naive | Naive | Naive 32 | Naive 32 | Naive 32 | Naive 1024 | Naive 1024 |

**Table 4.** Solving a linear system of size $r = 8$ with $p = 536871001$, in milliseconds.

| $n$ | 4 | 16 | 64 | 256 | 1024 |
|---|---|---|---|---|---|
| Newton | 930 | 2600 | 14000 | 140000 | 1300000 |
| Mmx | 3600 | 18000 | 53000 | 150000 | 1000000 |
| Variant | Naive | Naive | Naive | Naive 32 | Naive 32 |

**Table 5.** Solving a linear system of size $r = 128$ with $p = 536871001$, in milliseconds.

# Timings

Let us set $q = p^{2^j}$ where $p = 536871001$. We solve a linear system $\mathrm{B}\, C = A$ in $\mathbb{Q}$, with coefficients of $A$ and $\mathrm{B}$ in $\{0, ..., q-1\}$ using the rational reconstruction from $\mathbb{Z}_p$.

| $j$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| LinBox | 1.0 | 1.2 | 1.4 | 2.0 | 3.6 | 8.7 | 25 | 85 | 310 | 1200 | 4700 | 19000 | 77000 | 310000 | $\infty$ |
| Mmx | 0.10 | 0.14 | 0.24 | 0.43 | 0.58 | 0.96 | 2.1 | 5.3 | 14 | 34 | 90 | 250 | 490 | 1100 | 3100 |

**Table 6.** Solving a system of size 4, in milliseconds.

| $j$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| LinBox | 2.3 | 3.6 | 7.0 | 16 | 40 | 120 | 410 | 1500 | 5700 | 23000 | 91000 |
| Mmx | 2.9 | 5.6 | 13 | 20 | 30 | 56 | 140 | 380 | 1000 | 2800 | 7400 |

**Table 7.** Solving a system of size 16, in milliseconds.

| $j$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| LinBox | 5.9 | 11 | 25 | 61 | 170 | 540 | 1900 | 7000 | 27000 | 110000 | 480000 |
| Mmx | 24 | 54 | 150 | 200 | 360 | 780 | 2000 | 5500 | 14000 | 37000 | 92000 |

**Table 8.** Solving a system of size 32, in milliseconds.

# Regular root lifting of dense multivariate algebraic systems in $\mathbb{Z}_p$

**Proposition 18.** *Let $\boldsymbol{P}$ be an algebraic system in $\mathbb{Z}[\boldsymbol{Y}]^r$, where $\boldsymbol{Y} = (Y_1, ..., Y_r)$ and let $\boldsymbol{y}_0$ be a regular root of $\boldsymbol{P}$ modulo $p$. Then, there exists a unique $\boldsymbol{y} \in \mathbb{Z}_p^r$ such that $\boldsymbol{y} = \boldsymbol{y}_0 + O(p)$ and $\boldsymbol{P}(\boldsymbol{y}) = 0$ and it is a fixed-point of*

$$\boldsymbol{\Phi}(\boldsymbol{Y}) = \mathrm{d}\,\boldsymbol{P}_{\boldsymbol{y}_0}^{-1}(\mathrm{d}\,\boldsymbol{P}_{\boldsymbol{y}_0}(\boldsymbol{Y}) - \boldsymbol{P}(\boldsymbol{Y})).$$

*Using the following algorithm, one can compute $\boldsymbol{y}$ at precision $n$ in $r\,d^r\,\mathsf{R}(n)$ bit-operations, where $d = \deg \boldsymbol{P}$.*

# Regular root lifting of dense multivariate algebraic systems in $\mathbb{Z}_p$

For $j, k$ such that $1 \leq j \leq k \leq r$, let $\boldsymbol{Q}^{(j,k)}$ be algebraic systems such that

$$\boldsymbol{P}(\boldsymbol{Y}) = \boldsymbol{P}(\boldsymbol{y}_0) + \mathrm{d}\,\boldsymbol{P}(\boldsymbol{y}_0)(\boldsymbol{Y}) + \sum_{1 \leq j \leq k \leq r} \boldsymbol{Q}^{(j,k)}(\boldsymbol{Y})\,(Y_j - y_{j,0})\,(Y_k - y_{k,0}).$$

**Algorithm 2**

**Input.** $\boldsymbol{P} \in \mathbb{Z}[\boldsymbol{Y}]^r$ with a regular root $\boldsymbol{y}_0$ in $\mathbb{F}_p^r$.
**Output.** A shifted algorithm $\boldsymbol{\Psi}$ associated to $\boldsymbol{\Phi}$ and $\boldsymbol{y}_0$.

1. For $1 \leq j \leq k \leq r$, compute $\boldsymbol{Q}^{(j,k)}(\boldsymbol{Y})$ from $\boldsymbol{P}(\boldsymbol{Y})$.
2. For $1 \leq j \leq k \leq r$, let $\mathrm{pr}_{j,k}(\boldsymbol{Z})$: $\boldsymbol{Z} \mapsto \left(\frac{Z_j - y_{j,0}}{p}\right)\left(\frac{Z_k - y_{k,0}}{p}\right)$.
3. **Return** the shifted algorithm

$$\boldsymbol{\Psi}\colon\ \boldsymbol{Z} \mapsto \mathrm{d}\,\boldsymbol{P}_{\boldsymbol{y}_0}^{-1}\left(-\boldsymbol{P}(\boldsymbol{y}_0) + \mathrm{d}\,\boldsymbol{P}_{\boldsymbol{y}_0}(\boldsymbol{y}_0) - p^2\left(\sum_{1 \leq j \leq k \leq r} \boldsymbol{Q}^{(j,k)}(\boldsymbol{Z}) \cdot \mathrm{pr}_{j,k}(\boldsymbol{Z})\right)\right).$$

# Regular root lifting of s.l.p. multivariate algebraic systems in $\mathbb{Z}_p$

For an algebraic system $\boldsymbol{P}(\boldsymbol{Y})$, we define algebraic systems $\boldsymbol{T_P}(\boldsymbol{Y}) = \boldsymbol{P}(\boldsymbol{y}_0) + \mathrm{d}\,\boldsymbol{P}_{\boldsymbol{y}_0}(\boldsymbol{Y} - \boldsymbol{y}_0)$ and $\boldsymbol{E_P}(\boldsymbol{Y}) = \boldsymbol{P}(\boldsymbol{Y}) - \boldsymbol{T_P}(\boldsymbol{Y})$.

**Proposition 19.** *If $\boldsymbol{P}$ is given as a s.l.p. with $L^*$ multiplications $\cdot$, we define recursively vectors $\boldsymbol{\tau}_j \in \mathbb{Z}^{r+1}$ and s.l.p.'s $\boldsymbol{\varepsilon}_j$ for $1 \le j \le r$. Initially, $\boldsymbol{\varepsilon}_j^{-i+r} := 0$ and $\boldsymbol{\tau}_j^{-i+r} := (y_{i,0}, 0, ..., 0, 1, 0, ..., 0)$ with $1$ at index $i+1$, then we define recursively $\boldsymbol{\varepsilon}_j^i$ and $\boldsymbol{\tau}_j^i$ as before.*

*Let $\boldsymbol{\tau}_j^u = (a_0, a_1, ..., a_r)$ and $\boldsymbol{\tau}_j^v = (b_0, b_1, ..., b_r)$. If $\Gamma_i(\cdot, u, v)$, then*

$$\begin{aligned}
\boldsymbol{\tau}_j^i &= (a_0\,b_0, a_0\,b_1 + a_1\,b_0, ..., a_0\,b_r + a_r\,b_0) \\
\boldsymbol{\varepsilon}_j^i &= \boldsymbol{\varepsilon}_j^u \cdot \boldsymbol{\varepsilon}_j^v + p\left(\left(\sum_{\ell=1}^{r} a_\ell\,(Z_\ell - y_{0,\ell})\right) \cdot \frac{\boldsymbol{\varepsilon}_j^v}{p} + \left(\sum_{\ell=1}^{r} b_\ell\,(Z_\ell - y_{0,\ell})\right) \cdot \frac{\boldsymbol{\varepsilon}_j^u}{p}\right) \\
&\quad + p^2\left(\sum_{1 \le \ell_1, \ell_2 \le r} a_{\ell_1}\,b_{\ell_2}\left(\frac{Z_{\ell_1} - y_{0,\ell_1}}{p} \cdot \frac{Z_{\ell_2} - y_{0,\ell_2}}{p}\right)\right).
\end{aligned}$$

*If each $P_j$ has $k_j$ instructions, we let $\boldsymbol{\tau_P} = \left(\tau_1^{k_1}, ..., \tau_r^{k_r}\right)$ and $\boldsymbol{\varepsilon_P} = \left(\varepsilon_1^{k_1}, ..., \varepsilon_r^{k_r}\right)$. Furthermore, $\boldsymbol{\varepsilon_P}$ is a shifted algorithm for $\boldsymbol{E_p}$ and $\boldsymbol{y}_0$ with $3\,L^* + \frac{r\,(r+1)}{2}$ multiplications. Moreover, $\tau_j^{k_j}$ is the vector of coefficients of $T_{P_j}$ in the basis $(1, Y_1 - y_{0,1}, ..., Y_r - y_{0,r})$.*

# Regular root lifting of s.l.p. multivariate algebraic sytems in $\mathbb{Z}_p$

**Proposition 20.** *Let $\boldsymbol{P}(\boldsymbol{Y}) \in \mathbb{Z}_p[\boldsymbol{Y}]^r$ be a multivariate algebraic system given as a s.l.p. with $L^*$ multiplications between $p$-adic integers. Let $\boldsymbol{y}_0$ be a regular root of $\boldsymbol{P}$ in $\mathbb{F}_p^r$. Then, the following algorithm:*

$$\boldsymbol{\Psi}\colon\ \boldsymbol{Z} \mapsto \mathrm{d}\,\boldsymbol{P}_{\boldsymbol{y}_0}^{-1}(-\boldsymbol{P}(\boldsymbol{y}_0) + \mathrm{d}\,\boldsymbol{P}_{\boldsymbol{y}_0}(y_0) - \boldsymbol{\varepsilon}_{\boldsymbol{P}}(\boldsymbol{Z}))$$

*is a shifted algorithm associated to $\boldsymbol{\Phi}$ and $\boldsymbol{y}_0$ whose multiplicative complexity is $3\,L^* + \frac{r\,(r+1)}{2}$.*

***Theorem* 21.** *The root $\boldsymbol{y} \in \mathbb{Z}_p^r$ of $\boldsymbol{P}$ can be computed at precision $n$ in $\left(3\,L^* + \frac{r\,(r+1)}{2}\right)\mathsf{R}(n) + O(n)$ bit-operations.*

Thank you

for your attention!