Linear algebra over p-adics

This chapter deals with the resolution of linear systems over the p-adics. Linear algebra problems are often classified into broad categories, depending on whether the matrix of the system is dense, sparse, structured, ... In the context of solving over the p-adics, most previous algorithms rely on lifting techniques using either Dixon's / Moenck-Carter's algorithm, or Newton iteration, and can to some extent exploit the structure of the given matrix.

In this chapter, we introduce an algorithm based on the p-recursive framework of Chapter 2, which can in principle be applied to all above families of matrices. We will focus on two important cases, *dense* and *structured* matrices, and show how our algorithm can improve on existing techniques in these cases.

The relaxed linear system solver applied to dense matrices is a common work with J. Berthomieu, published as a part of [BL12]. The application to structured matrices is a joint work in progress with É. Schost.

3.1 Overview

Assumptions on the base ring Throughout this chapter, we continue using some notation and assumptions introduced in Chapter 1: R is our base ring (typically, $\mathbb{Z}$ or $\Bbbk[X]$), p is a non-zero element in R (typically, a prime in $\mathbb{Z}$ or $X \in \Bbbk[X]$) and R_p is the completion of R for the p-adic topology (so we get for instance the p-adic integers, or the power series ring $\Bbbk[[X]]$). In order to simplify some considerations below regarding the notion of rank of a matrix over a ring, we will make the following assumption in all this chapter: *both R and R_p are domains*; this is the case in the examples above.

As before, we fix a set M of representatives of $R/(p)$, which allows us to define the *length* $\lambda(a)$ of a non zero p-adic $a \in R_p$; recall that we make the assumption that the elements of $R \subset R_p$ have finite length. We generalize the length function to vectors or matrices of p-adics by setting $\lambda(A) := \max_{1 \leqslant i \leqslant r, 1 \leqslant j \leqslant s} (\lambda(A_{i,j}))$ if $A \in \mathcal{M}_{r \times s}(R_p)$.

Problem statement We consider a linear system of the form $A = B \cdot C$, where A and B are known, and C is the unknown. The matrix A belongs to $\mathcal{M}_{r\times s}(R_p)$ and $B \in \mathcal{M}_{r\times r}(R_p)$ is invertible; we solve the linear system $A = B \cdot C$ for $C \in \mathcal{M}_{r\times s}(R_p)$. We make the natural assumption that $s \leqslant r$; the most interesting cases are $s = 1$ (which amounts to linear system solving) and $s = r$, which contains in particular the problem of inverting B (our algorithm handles both cases in a uniform manner).

A major application of p-adic linear system solving is actually to solve systems over R (in the two contexts above, this means systems with integer, resp. polynomial coefficients), by means of lifting techniques (the paper [MC79] introduced this idea in the case of integer linear systems). In such cases, the solution C belongs to $\mathcal{M}_{r\times s}(Q)$, where Q is the fraction field of R, with a denominator invertible modulo p. Using p-adic techniques, we can compute the expansion of C in $\mathcal{M}_{r\times s}(R_p)$, from which C itself can be reconstructed by means of rational reconstruction — we will focus on the lifting step, and we will not detail the reconstruction step here.

In order to describe such situations quantitatively, we will use the following parameters: the length of the entries of A and B, that is, $d := \max(\lambda(A), \lambda(B))$, and the precision N to which we require C; thus, we will always be able to suppose that $d \leqslant N$. The case $N = d$ corresponds to the resolution of p-adic linear systems proper, whereas solving systems over R often requires to take a precision $N \gg d$. Indeed, in that case, we deduce from Cramer's formulas that the numerators and denominators of C have length $\mathcal{O}(r\,(d + \log(r)))$, so that we take N of order $\mathcal{O}(r\,(d + \log(r)))$ in order to make rational reconstruction possible.

For computations with structured matrices, we will use a different, non-trivial representation for B, by means of its "generators"; then, we will denote by d' the length of these generators. Details are given below.

Complexity model Throughout this chapter, we represent all p-adics through their base-M expansion, and we measure the cost of an algorithm by the number of arithmetic operations on p-adics of length 1 (*i.e.* with only a constant coefficient) it performs, as explained in Chapter 1.

The algorithms in this chapter will rely on the notion of *shifted decomposition*: a shifted decomposition of a p-adic $a \in R_p$ is simply a pair $(\sigma_a, \delta_a) \in R_p^2$ such that $a = \sigma_a + p\,\delta_a$. A simple particular case is $(a \bmod p, a \text{ quo } p)$; this is by no means the only choice. This notion carries over to matrices without difficulty.

We denote by $\mathsf{I}(N)$ the cost of multiplication of two p-adics at precision N and we let $\mathsf{R}(N)$ be the cost of multiplying two p-adics at precision N by an on-line algorithm. As in Chapter 1, we let further $\mathsf{M}(d)$ denote the arithmetic complexity of multiplication of polynomials of degree at most d over any ring (we will need this operation for the multiplication of structured matrices). Remark that when $R = \Bbbk[X]$, I and M are the same thing, but this may not be the case anymore over other rings, such as $\mathbb{Z}$.

Let next $\mathsf{I}(r, d)$ be the cost of multiplying two polynomials in $R_p[Y]$ with degree at most r and coefficients of length at most d. Since the coefficients of the product polynomial have length at most $2\,d + \lceil \log_2(r) \rceil$, we deduce that we can take

$$\mathsf{I}(r, d) = \mathcal{O}(\mathsf{M}(r)\ \mathsf{I}(d + \log(r)))$$

by working modulo p to the power the required precision; over $R_p = \Bbbk[[X]]$, the $\log(r)$ term vanishes since no carry occurs.

Let us focus on the corresponding on-line algorithm. We consider these polynomials as p-adics of polynomials, *i.e.* p-adic whose coefficients are polynomials in M. We denote by $\mathsf{R}(r, N)$ the cost of an on-line multiplication at precision N of polynomials of degrees at most r. As in Chapter 1, this cost is bounded by $\mathsf{R}(r, N) = \mathcal{O}(\mathsf{I}(r, N)\log(N))$ in the case of power series rings or p-adic integers. If the length d' of the coefficients of one operand is less than N, the cost reduces to $\mathcal{O}(N\,\mathsf{R}(r, d')/d')$.

Now, let us turn to matrix arithmetic. We let ω be such that we can multiply $r \times r$ matrices within $\mathcal{O}(r^\omega)$ ring operations over any ring. The best known bound on ω is $\omega \leqslant 2.3727$ [CW90, Sto10, VW11]. It is known that, if the base ring is a field, we can invert any invertible matrix in time $\mathcal{O}(r^\omega)$ base field operations. We will further denote by $\mathsf{MM}(r, s, d)$ the cost of multiplication of matrices A, B of sizes $(r \times r)$ by $(r \times s)$ over R_p, for inputs of length at most d. In our case $s \leqslant r$, and taking into account the growth of the length in the output, we obtain that $\mathsf{MM}(r, s, d)$ satisfies

$$\mathsf{MM}(r, s, d) = \mathcal{O}(r^2\, s^{\omega-2}\, \mathsf{I}(d + \log(r))),$$

since $\lambda(A \cdot B) \leqslant 2\,d + \lceil \log_2(r) \rceil$; the exponents on r and s are obtained by partitioning A and B into square blocks of size s.

Let us now consider the relaxed product of p-adic matrices, *i.e.* p-adic whose coefficients are matrices over M. We denote by $\mathsf{MMR}(r,\ s,\ N)$ the cost of the relaxed multiplication of a p-adic matrix of size $r \times r$ by a p-adic matrix of size $r \times s$ at precision N. As in Chapter 1, we can connect the cost of off-line and on-line multiplication algorithms by

$$\mathsf{MMR}(r, s, N) = \mathcal{O}(\mathsf{MM}(r, s, N)\log(N))$$

in the case of power series rings or p-adic integers. Likewise, we also notice that the relaxed multiplication of two matrices $A, B \in (\mathcal{M}_{r\times s}(R))_{(p)}$ at precision N with $d := \lambda(A) \leqslant N$ takes time $\mathcal{O}(N\,\mathsf{MMR}(r, s, d)/d)$.

Previous work The first algorithm we will mention is due to Dixon [Dix82]; it finds one p-adic coefficient of the solution C at a time and then updates the matrix A. On the other side of the spectrum, one finds Newton's iteration, which doubles the precision of the solution at each step (and can thus benefit from fast p-adic multiplication); however, this algorithm computes the whole inverse of B at precision N, which can be too costly when we only want one vector solution.

Moenck-Carter's algorithm [MC79] is a variant of Dixon's algorithm that works with p^ℓ-adics instead of p-adics. It takes advantages of fast truncated p-adic multiplication but requires that we compute the inverse of B at precision d (for which Newton iteration is used).

Finally, Storjohann's high-order lifting algorithm [Sto03] can be seen as a fast version of Moenck-Carter's algorithm, well-suited to cases where $d \ll N$. That algorithm was presented for $R = \Bbbk[X]$ and the result was extended to the integer case in [Sto05]. We believe that the result could carry over to any p-adic ring.

Historically, these algorithms were all introduced for dense matrices; however, most of them can be adapted to work with structured matrices. The exception is Storjohann's high-order lifting, which does not seem to carry over in a straightforward manner.

Main results The core of this chapter is an algorithm to solve linear systems by means of relaxed techniques; it is obtained by proving that the entries of the solution $C = B^{-1} \cdot A$ are p-recursive. In other words, we show that C is a fixed point for a suitable shifted operator.

This principle can be put to use for several families of matrices; we detail it for dense and structured matrices. Taking for instance $s = 1$, to compute C at precision N, the cost of the resulting algorithm will (roughly speaking) involve the following:

- the inversion of B modulo (p),
- $\mathcal{O}(N)$ matrix-vector products using the inverse of B modulo (p), with a right-hand side vector whose entries have length 1,
- $\mathcal{O}(1)$ matrix-vector product using B, with a right-hand side vector whose entries are relaxed p-adics.

Tables 3.1 and 3.2 give the resulting running time for the case of dense matrices, together with the results based on previous algorithms mentioned above; recall that $d = \lambda(B)$ and that N is the target precision. In the first table, we are in the general case $1 \leqslant s \leqslant r$; in the second one, we take $R = \Bbbk[X]$ and $s = 1$, and we choose two practically meaningful values for N, respectively $N = d$ and $N = r\,d$ (which was mentioned above). For the high-order lifting, the * indicates that the result is formally proved only for $R_p = \Bbbk[[X]]$ and $R = \mathbb{Z}$. The complexity $\mathsf{MM}(r, s\,N/d, 1)$ that appears in this case is bounded by $\mathsf{MM}(r, s\,N/d, 1) = r^{\omega-1}\,s\,N/d$.

Most previous complexity results are present in the literature, so we will not reprove them all; we only do it in cases where small difficulties may arise. For instance, Newton's algorithm and its cost analysis extend in a straightforward manner, since we only do computations modulo powers of p, which behave over general p-adics as they do over e.g. $R_p = \Bbbk[[X]]$; thus, we will not reprove the running time in this case. On the other hand, we will re-derive the cost of Dixon's and Moenck-Carter's algorithms, since they involve computations in R_p itself (*i.e.*, without reduction modulo a power of p), and considerations about the lengths of the operands play a role.

In most entries (especially in the first table), two components appear: the first one involves inverting the matrix B modulo (p), or a higher power of p and is independent of N; the second one describes the lifting process itself. In some cases, the cost of the first step can be neglected compared to the cost of the second one.

It appears in the last table that for solving up to precision $N = d$, our algorithm is the fastest among the ones we compare; for $N = r\,d$, Storjohann's high-order lifting does best (as it is specially designed for such large precisions).

Algorithm	Cost
Dixon	$\mathcal{O}(r^\omega + \mathsf{MM}(r,s,1)\ N\ d)$
Moenck-Carter	$\mathcal{O}\big(r^\omega\, \mathsf{I}(d) + \mathsf{MM}(r,s,d)\ \frac{N}{d}\big)$
Newton iteration	$\mathcal{O}(r^\omega\ \mathsf{I}(N))$
High-order lifting*	$\mathcal{O}\big(r^\omega \log(\frac{N}{d})\ \mathsf{I}(d) + \mathsf{MM}(r, s\,\frac{N}{d}, 1)\ \mathsf{I}(d)\big)$
Our algorithm	$\mathcal{O}\Big(r^\omega + N\ \frac{\mathsf{MMR}(r,s,d)}{d}\Big)$

Table 3.1. Cost of solving $A = B \cdot C$ for dense matrices

Algorithm	$N = d$	$N = r\,d$
Dixon	$\tilde{\mathcal{O}}(r^\omega + r^2\,d^2)$	$\tilde{\mathcal{O}}(r^3\,d^2)$
Moenck-Carter	$\tilde{\mathcal{O}}(r^\omega\,d)$	$\tilde{\mathcal{O}}(r^3\,d)$
Newton iteration	$\tilde{\mathcal{O}}(r^\omega\,d)$	$\tilde{\mathcal{O}}(r^{\omega+1}\,d)$
High-order lifting*	$\tilde{\mathcal{O}}(r^\omega\,d)$	$\tilde{\mathcal{O}}(r^\omega\,d)$
Our algorithm	$\tilde{\mathcal{O}}(r^\omega + r^2\,d)$	$\tilde{\mathcal{O}}(r^3\,d)$

Table 3.2. Simplified cost of solving $A = B \cdot C$ for dense matrices over $R_p = \mathbb{k}[[X]]$, with $s = 1$

Next, we discuss the situation for structured matrices; for that, a brief reminder is in order (for a thorough presentation, see [Pan01]).

A typical family of structured matrices are Toeplitz matrices, which are invariant along diagonals; exploiting this structure, one can multiply and invert such matrices in quasi-linear time. In this chapter, we will consider structured matrices as being matrices which are "close" to being Toeplitz. Formally, let us define the operator

$$\begin{array}{rccc} \phi_+\colon & \mathcal{M}_{r\times r}(R_p) & \to & \mathcal{M}_{r\times r}(R_p) \\ & A & \mapsto & A - A', \end{array}$$

where A' is obtained by shifting A down and right by one unit. If A is Toeplitz, $\phi_+(A)$ is zero, except in the first row and column; the key remark is that in this case, $\phi_+(A)$ has a small rank (at most 2), and can be written $\phi_+(A) = G \cdot H^t$, with G and H matrices of sizes $r \times 2$, with entries in R_p.

The key idea is then to measure the "structure" of the matrix A as the rank of $\phi_+(A)$, which is called its *displacement rank*, usually denoted by $\alpha(A)$. If $\alpha(A) \leqslant \alpha$, then there exist matrices G and H in $\mathcal{M}_{r\times\alpha}(R_p)$ such that $\phi_+(A) = G \cdot H^t$.

The key idea of algorithms for structured matrices is to use such generators as a compact data structure to represent A, since they can encode A using $\mathcal{O}(\alpha\ r)$ elements of R_p instead of r^2. As typical examples, note that the displacement rank

of a Sylvester matrix is at most 2; more generally, matrices coming from e.g. Padé-Hermite approximation problems have small displacement ranks. The last important property of structured matrices is that the matrix-vector multiplication $A \cdot V$ for $A \in \mathcal{M}_{r\times r}(R_p)$ and $V \in \mathcal{M}_{r\times 1}(R_p)$ boils down to polynomial multiplication, so that it costs $\mathcal{O}(\alpha\,\mathsf{M}(r))$ (we will also need to pay attention to the precision of the arguments).

Table 3.3 recalls previously known results about solving structured linear systems and shows the running time of our algorithm. Recall that here, d' denotes the length of the generators of B and N is still the target precision. As before, Table 3.4 gives simplified results for $s=1$ and $N=d'$ and $N=r\,d'$.

Previous algorithms can all be found in [Pan01] for rings such as $R=\mathbb{Z}$ and $R=\Bbbk[X]$, so as in the case of dense matrices, we will only prove those cost estimates where attention must be paid to issues such as the length of the p-adics. Note that the high-order lifting entry has disappeared, since we do not know how to extend it to the structured case. As in the dense case, the running times involve two components: inverting the matrix modulo (p), then the lifting itself.

Algorithm	Cost
Dixon	$\mathcal{O}(\alpha^2\,\mathsf{M}(r)\log(r)+\alpha\,s\,\mathsf{M}(r)\,N\,d')$
Moenck-Carter	$\mathcal{O}\Big(\alpha^2\,\mathsf{M}(r)\log(r)+\alpha^2\,\mathsf{M}(r)\,\mathsf{I}(d')+\alpha\,s\,N\,\frac{\mathsf{I}(r,\,d')}{d'}\Big)$
Newton iteration	$\mathcal{O}(\alpha^2\,\mathsf{M}(r)\log(r)+\alpha^2\,\mathsf{M}(r)\,\mathsf{I}(N)+\alpha\,s\,\mathsf{M}(r)\,\mathsf{I}(N))$
Our algorithm	$\mathcal{O}\Big(\alpha^2\,\mathsf{M}(r)\log(r)+\alpha\,s\,N\,\frac{\mathsf{R}(r,\,d')}{d'}\Big)$

Table 3.3. Cost of solving $A=B\cdot C$ for structured matrices

Algorithm	$N=d'$	$N=r\,d'$
Dixon	$\tilde{\mathcal{O}}\big(\alpha^2\,r+\alpha\,r\,d'^2\big)$	$\tilde{\mathcal{O}}\big(\alpha\,r^2\,d'^2\big)$
Moenck-Carter	$\tilde{\mathcal{O}}(\alpha^2\,r\,d')$	$\tilde{\mathcal{O}}(\alpha\,r^2\,d')$
Newton iteration	$\tilde{\mathcal{O}}(\alpha^2\,r\,d')$	$\tilde{\mathcal{O}}(\alpha^2\,r^2\,d')$
Our algorithm	$\tilde{\mathcal{O}}(\alpha^2\,r+\alpha\,r\,d')$	$\tilde{\mathcal{O}}(\alpha\,r^2\,d')$

Table 3.4. Simplified cost of solving $A=B\cdot C$ for structured matrices over $\Bbbk[[X]]$, with $s=1$

To summarize, in all these cases, our algorithm performs at least as well, and often better, than previous algorithms.

The relaxed linear system solver applied to dense matrices was published as a part of [BL12]. It has been implemented inside the MATHEMAGIX computer algebra system [HLM+02]. The application to structured matrices is a joint work in progress with É. SCHOST.

3.2 Structured matrices

While we need only fairly standard results about dense matrices, we believe it is worth recalling a few facts about the structured case. We will need very little of the existing results on structured matrices: mainly, how to reconstruct a matrix from its generators, as well as a few properties of the inverse of a structured matrix.

Let us start with a discussion of the inverse of A (assuming A is invertible). Then, is it know that the displacement rank $\alpha(A^{-1})$ is at most $\alpha(A)+2$, so that A^{-1} can be represented in a compact manner. What's more, generators of A^{-1} can be computed in time $\mathcal{O}(\alpha^2\,\mathsf{M}(r)\log(r))$ (using a Las-Vegas algorithm); this is called the Morf / Bitmead-Anderson algorithm [Mor74, Mor80, BA80].

At the heart of most algorithms for structured matrices lies the following question. Let G, H be generators for a matrix A. To use the generators G and H as a data structure, we must be able to recover A from these matrices. Indeed, the operator ϕ_+ is bijective, and it can be inverted as follows. Denote by H_i and G_i the columns of G and H, for $1 \leqslant i \leqslant \alpha$. For any $V = (v_0, \ldots, v_{r-1}) \in R_p^r$, we define the lower (resp. upper) triangular matrix $L(V)$ (resp. $U(V)$) by

$$L(V) := \begin{pmatrix} v_0 & 0 & \cdots & 0 \\ v_1 & v_0 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ v_{r-1} & \cdots & v_1 & v_0 \end{pmatrix} \in \mathcal{M}_{r\times r}(R_p)$$

and $U(V) := L(V)^t$. Then, for any matrices $G, H \in \mathcal{M}_{r\times\alpha}$, we have the equivalence

$$\phi_+(A) = G\cdot H^t \quad\Leftrightarrow\quad A = \sum_{i=1}^{\alpha} L(G_i)\cdot U(H_i).$$

This representation of A is essential to perform the matrix-vector multiplication by A efficiently. For $n \in \mathbb{N}$ and $P = \sum_{i=0}^{n} P_i\, Y^i \in R_p[Y]$, we denote by $\mathrm{rev}_n(P)$ the reverse polynomial of P defined by

$$\mathrm{rev}_n(P) := \sum_{i=0}^{n} P_{n-i}\, Y^i \in R_p[Y]_{\leqslant n}.$$

Besides, to a vector $V := [v_0, \ldots, v_{r-1}]^t \in \mathcal{M}_{r\times 1}(R_p)$, we associate the polynomial $v \in R_p[X]$ defined by $v := \sum_{i=0}^{r-1} v_i\, X^i$. This association is bijective. Next, if a, c, $v \in R_p[Y]_{<r}$ are the polynomials associated to some vectors $A, C, V \in \mathcal{M}_{r\times 1}(R)$, then

$$c = \begin{cases} \mathrm{rev}_{r-1}(a\,\mathrm{rev}_{r-1}(v)) & \text{if } C = U(A)\cdot V \\ a\,v \bmod Y^r & \text{if } C = L(A)\cdot V. \end{cases} \tag{3.1}$$

This shows that given generators for A of size $r\times\alpha$, the matrix-vector multiplications $A\cdot V$ can be done using $2\,\alpha$ short products of polynomials in degree r.

To estimate costs precisely, we have to take into account the size of the operands. We will thus consider the length d' of the entries of the displacement generators. Then, if a vector V has length d' as well, the matrix-vector multiplication $B\cdot V$ costs $\mathcal{O}(\alpha\,\mathsf{I}(r,d'))$, with $\alpha := \alpha(B)$. Indeed, the cost of all multiplications is easily seen to be controlled by the above bound; the cost of additions follows from Lemma 1.1.

We can further deduce an on-line algorithm for the matrix-vector multiplication $B \cdot V$. For the polynomial multiplication of $R_p[Y]$ of Equation (3.1), use on-line algorithms on p-adic of polynomials. This algorithm is on-line with respect to the entries of the displacement generators G, H of B and with respect to V. It computes $B \cdot V$ at precision N in time $\mathcal{O}(\alpha\, \mathsf{R}(r, N))$. If the length $d' := \max(\lambda(G), \lambda(H))$ of the displacement generators is less than N, then the on-line multiplication $B \cdot V$ takes time $\mathcal{O}(\alpha\, N\, \mathsf{R}(r, d')/d')$.

Since in many situations we will encounter, the matrix B is known, we can adapt the latter algorithm to be half-line, that is off-line in B and on-line in V. For this matter, just replace on-line multiplication algorithms in R_p by half-line algorithms (which are slightly cheaper, see Chapter 1).

3.3 Solving linear systems

In this section, we give the details of the algorithm underlying the new results in Tables 3.1 to 3.4. We start by recalling Dixon's algorithm (and briefly mention the closely related Moenck-Carter's algorithm). In the second subsection, we will show how this algorithm can be seen as a relaxed algorithm that computes C as a fixed point and is useful when B has small length; finally, the last subsection introduces the general algorithm.

As a preamble, note that any matrix $A \in \mathcal{M}_{r\times s}(R_p)$ can be seen as a p-adic matrix, *i.e.* a p-adic whose coefficients are matrices over M. In this case, the coefficient matrix of index n will be denoted by $A_n \in \mathcal{M}_{r\times s}(M)$, so that $A = \sum_{n=0}^{\infty} A_n\, p^n$. We will use this notation frequently.

In this section, we denote by $d := \max(\lambda(A), \lambda(B))$ the maximum length of entries of A and B. If we assume that B is structured, its displacement rank is $\alpha := \alpha(B)$; in that case, as input, we assume that we are given generators G, H for B with entries in R_p, and we let $d' := \max(\lambda(G), \lambda(H))$. In all cases, we want C at precision N, so we suppose that $d, d' \leqslant N$.

3.3.1 Dixon's and Moenck-Carter's algorithms

The paper [Dix82] presented a simple algorithm to solve an integer linear system via a p-adic lifting; we present here a straightforward extension to our slightly more general context. This algorithm is based on the following lemma.

Lemma 3.1. *Let $B \in \mathcal{M}_{r\times r}(R_p)$ invertible and $A, C \in \mathcal{M}_{r\times s}(R_p)$ such that $A = B \cdot C$. Then for all $i \in \mathbb{N}$, there exists $A^{(i)} \in \mathcal{M}_{r\times s}(R_p)$ such that*

$$C = B^{-1} \cdot A = C_0 + C_1\, p + \cdots + C_{i-1}\, p^{i-1} + p^i\, B^{-1} \cdot A^{(i)}. \tag{3.2}$$

Proof. One has $A - B \cdot (C_0 + C_1\, p + \cdots + C_{i-1}\, p^{i-1}) = A - B \cdot C = 0$ in $\mathcal{M}_{r\times s}(R/(p^i))$. So we can define $A^{(i)} := p^{-i}\,[A - B \cdot (C_0 + C_1\, p + \cdots + C_{i-1}\, p^{i-1})]$ in $\mathcal{M}_{r\times s}(R_p)$ that satisfies Equation (3.2). □

The algorithm follows: at each step in the **for** loop, the matrix A is updated; the proof of the previous lemma shows that we are precisely computing the sequence $A^{(i)}$.

In order to analyze the algorithm, we need the following lemma describing the cost of polynomial or matrix multiplication with p-adic coefficients, in cases where the operands have unbalanced lengths. We will need the following extension of Lemma 1.1.

Lemma 3.2. *The following holds:*

- *Let P be in $\mathcal{M}_{r\times r}(R_p)$ and Q in $\mathcal{M}_{r\times s}(R_p)$, with $\lambda(P) = d$ and $\lambda(Q) = 1$. Then we can compute $S = P\,Q$ in time $\mathcal{O}(\mathsf{MM}(r,s,1)\,d)$.*
- *Let P, Q be in $R_p[X]_{<r}$ with $\lambda(P) = d$ and $\lambda(Q) = 1$. Then we can compute $S = P\,Q$ in time $\mathcal{O}(\mathsf{I}(r,1)\,d)$.*

Proof. In both cases, the strategy is the same. If the p-adic decomposition of Q is $\sum_{i=0}^{d-1} Q_i\,p^i$, then $S = \sum_{i=0}^{d-1} (PQ_i)\,p^i$. This amounts to d multiplications between operands of length 1 and some additions. Since the length of the entries of $P\,Q_i$ are bounded by $\lceil \log_2(r) \rceil$, Lemma 1.1 bounds the cost of the final addition by $\mathcal{O}(d \log(r))$, which is negligible compared to the cost of multiplications. □

Algorithm - `Dixon`
Input: $A \in \mathcal{M}_{r\times s}(R_p)$, $B \in \mathcal{M}_{r\times r}(R_p)$ and $N \in \mathbb{N}$ **Output:** $C \in \mathcal{M}_{r\times s}(R_p)$ such that $A = B \cdot C \bmod p^N$
1. $\Gamma = B^{-1} \bmod p$
2. $C_0 := (\Gamma \cdot A) \bmod p$
3. **for** i from 1 to $N-1$
a. $A := (A - B \cdot C_{i-1}) \operatorname{quo} p$
b. $C_i := (\Gamma \cdot A) \bmod p$
4. **return** $C := \sum_{i=0}^{N-1} C_i\,p^i$

Proposition 3.3. *Algorithm* `Dixon` *is correct and its cost is summed up in the table*

Dense matrices	$\mathcal{O}(r^\omega + \mathsf{MM}(r,s,1)\,N\,d)$
Structured matrices	$\mathcal{O}(\alpha^2\,\mathsf{M}(r)\log(r) + \alpha\,s\,\mathsf{M}(r)\,N d')$

Table 3.5. *Cost of Dixon's algorithm depending on matrix representation*

Proof. We refer to [Dix82] for the proof of correctness of the algorithm (which readily follows from the previous lemma).

After computing Γ, at each step, the algorithm performs one multiplication $B \cdot C_{i-1}$ and one multiplication $\Gamma \cdot A$ modulo p, plus some additions, remainders and quotients whose cost is dominated by the multiplications.

Let us first study the cost for dense matrices. Computing Γ takes $\mathcal{O}(r^\omega)$ operations (since this is arithmetic modulo (p)). To compute $B \cdot C_{i-1}$, we apply the previous lemma, since B has length d and C_{i-1} has length 1; the cost is $\mathcal{O}(\mathsf{MM}(r, s, 1)\, d)$ using Lemma 3.2. Computing $\Gamma \cdot A \bmod p$ is cheaper, since we do all operations modulo p. Taking all i into account, we get the claimed result.

For structured matrices, notice that $B \bmod p$ is a structured matrix of rank at most α and so $\Gamma = B^{-1} \bmod p$ has rank $\alpha(\Gamma) \leqslant \alpha + 2$. Therefore the cost for structured matrices is $\mathcal{O}(\alpha^2\, \mathsf{M}(r) \log(r))$, for the computation of Γ, plus, for each i, the cost induced by the products $B \cdot C_{i-1}$ and $\Gamma \cdot A \bmod p$. The latter is negligible. The former is done by means of $\mathcal{O}(\alpha\ s)$ polynomial multiplications in degree r, with coefficients of lengths respectively d' and 1. The cost estimate follows from the previous lemma. □

Moenck-Carter's algorithm can be seen as a p^ℓ-adic variant of Dixon's, where we compute ℓ p-adic coefficients of C at a time (thus, the algorithm is formally the same, up to replacing p by p^ℓ). By suitably choosing ℓ, it allows us to benefit from fast p-adics multiplication algorithms.

One quickly sees that the optimal asymptotic cost in N is obtained by choosing $\ell = d$ (in the dense case) and $\ell = d'$ (in the structured case). This gives the costs reported in Table 3.6 below, which we justify now.

Dense matrices	$\mathcal{O}\big(r^\omega\, \mathsf{I}(d) + \mathsf{MM}(r, s, d)\, \frac{N}{d}\big)$
Structured matrices	$\mathcal{O}\big(\alpha^2\, \mathsf{M}(r) \log(r) + \alpha^2\, \mathsf{M}(r)\, \mathsf{I}(d') + \alpha\, s\, \mathsf{M}(r, d')\, \frac{N}{d'}\big)$

Table 3.6. Cost of Moenck-Carter's algorithm depending on matrix representation

The algorithm starts by computing the inverse Γ of B modulo p^ℓ; for this, we use Newton iteration, whose cost was recalled in the previous section. At each step of the loop, the algorithm computes $\Gamma \cdot A$ modulo p^ℓ and $B \cdot C_i$, where now C_i has length ℓ.

For dense matrices, taking $\ell = d$, the product $\Gamma \cdot A$ modulo p^ℓ takes time $\mathcal{O}(r^2\, s^{\omega-2}\, \mathsf{I}(d))$, which is always less than the cost of $B \cdot C_i$, that is $\mathcal{O}(\mathsf{MM}(r, s, d))$. Since the loop now has length N/d, it sums to the announced cost.

For structured matrices, we take $\ell = d'$. Using Newton iteration., the first inversion costs $\mathcal{O}(\alpha^2\, \mathsf{M}(r) \log(r) + \alpha^2\, \mathsf{M}(r)\, \mathsf{I}(d'))$. For each i in the main loop, the product $\Gamma \cdot A$ modulo p^ℓ takes time $\mathcal{O}(\alpha\, s\, \mathsf{M}(r)\, \mathsf{I}(d'))$, which is always less than the cost of computing $B \cdot C_i$, that is $\mathcal{O}(\alpha\, s\, \mathsf{M}(r, d'))$. Since we now do N/d' passes through the loop, this gives the announced cost.

3.3.2 The on-line point of view on Dixon's algorithm

We published in [BL12, Section 4.1] an on-line algorithm to solve linear systems $B \cdot C = A$ for C, well-adapted to cases where $\lambda(B)$ is small. With hindsight, we realized that this algorithm coincides with Dixon's algorithm.

In this subsection, we make this remark more precise: we prove that Dixon's algorithm is on-line, by presenting it slightly differently to write it as a fixed point algorithm `OnlineDixon`. Then we prove that this algorithm is a shifted algorithm. This will be useful for the next subsection, where we deal with cases where $\lambda(B)$ is arbitrary.

We will use two operators `Mul_rem` and `Mul_quo`, defined for $B \in \mathcal{M}_{r\times r}(R_p)$ and $A \in \mathcal{M}_{r\times s}(R_p)$ by

$$\begin{aligned} \texttt{Mul\_rem}(B,A) &:= \sum_{n\in\mathbb{N}} (B\cdot A_n \bmod p)\, p^n \in \mathcal{M}_{r\times s}(R_p) \\ \texttt{Mul\_quo}(B,A) &:= \sum_{n\in\mathbb{N}} (B\cdot A_n \operatorname{quo} p)\, p^n \in \mathcal{M}_{r\times s}(R_p) \end{aligned}$$

so that we have

$$B\cdot A = \texttt{Mul\_rem}(B,A) + p\,\texttt{Mul\_quo}(B,A).$$

We see on these definitions that `Mul_rem`$(B,\ A)$ and `Mul_quo`$(B,\ A)$ are on-line algorithms with respect to the input A.

Proposition 3.4. *Let $A \in \mathcal{M}_{r\times s}(R_p)$ and $B \in \mathcal{M}_{r\times r}(R_p)$. Suppose B is invertible modulo p and define $\Gamma := B^{-1} \bmod p$. Set $C_0 := (\Gamma\cdot A) \bmod p$ and let* `OnlineDixon` *denote the algorithm*

$$\texttt{OnlineDixon}(A,B,Y) := \texttt{Mul\_rem}(\Gamma, A - p\times \texttt{Mul\_quo}(B,Y)).$$

Then, Algorithm `OnlineDixon` *has shift 1 with respect to its input Y and has shift 0 with respect to its input A. Moreover,*

$$C = \texttt{OnlineDixon}(A,B,C).$$

Proof. First,

$$\begin{aligned} & & A &= \texttt{Mul\_rem}(B,C) + p\,\texttt{Mul\_quo}(B,C) \\ \Rightarrow & & \texttt{Mul\_rem}(B,C) &= A - p\,\texttt{Mul\_quo}(B,C) \\ \Rightarrow & & C &= \texttt{Mul\_rem}(\Gamma, A - p\,\texttt{Mul\_quo}(B,C)) \end{aligned}$$

so that $\Psi(C) = C$.

Next, for any $n \in \mathbb{N}$, the p-adic coefficient $\texttt{OnlineDixon}(A,B,C)_n$ requires the nth p-adic coefficient of $A - p\times \texttt{Mul\_quo}(B,C)$. This computation reads at most the coefficients A_i with $0 \leqslant i \leqslant n$, so $0 \in \mathcal{S}(\texttt{OnlineDixon}, 1)$ (see Definition 2.8). It also requires the coefficient $\texttt{Mul\_quo}(B,C)_{n-1}$, which reads at most the coefficients C_i with $0 \leqslant i \leqslant n-1$, so $1 \in \mathcal{S}(\texttt{OnlineDixon}, 3)$. □

Dixon's algorithm coincides with Algorithm `OnlineDixon`. Indeed, in the on-line algorithm presented here, the computation $A - p\ \texttt{Mul\_quo}(B,\ C)$ subtracts $\texttt{quo}(B\cdot C_i, p)$ to A at step i, which corresponds to the instruction

$$A := (A - B\cdot C_i) \operatorname{quo} p$$

in Dixon's algorithm. Similarly, the `Mul_rem` operations corresponds to the computation modulo p in Dixon's algorithm.

3.3.3 On-line solving of p-adic linear systems

For dense matrices, the running time analysis showed that Dixon's algorithm is satisfactory when $\lambda(B) = 1$, but not anymore when $\lambda(B)$ is large (since when $\lambda(B) \simeq N$, the behavior is then quadratic in N). The same holds for structured matrices — in that case, it is the length of the given generators that matters.

To by-pass this issue, we give our main result concerning the resolution of linear systems, which shows that the solution C of the system is a fixed point for an operator easily deduced from the original system, and whose matrix has better length properties than B. This is an extension of the algorithm for division of p-adics of [Hoe02, BHL11].

Proposition 3.5. *Let $A \in \mathcal{M}_{r\times s}(R_p)$ and $B \in \mathcal{M}_{r\times r}(R_p)$ be two matrices such that B_0 is invertible of inverse $\Gamma = B_0^{-1} \bmod p$. Let $C := B^{-1} \cdot A$ and $C_0 := (\Gamma \cdot A) \bmod p$. Let finally $(\sigma_B, \delta_B) \in \mathcal{M}_{r\times r}(R_p)^2$ be any shifted decomposition of B. We note $\texttt{OnlineSolve}(A, B, Y)$ the algorithm defined by*

$$\texttt{OnlineSolve}(A, B, Y) := \texttt{OnlineDixon}(A - p \times (\delta_B \cdot Y), \sigma_B, Y).$$

Then, the s.l.p. Ψ with operations in $\{+, -, \cdot, p^s \times \_, \_/p^s, \texttt{Mul\_rem}, \texttt{Mul\_quo}\} \cup R \cup R^c$ defined by

$$\Psi\colon \quad Y \longmapsto \texttt{OnlineSolve}(A, B, Y) \tag{3.3}$$

verifies that $\texttt{OnlineEvaluation}(\Psi, \_, N)$ has shift 1 and that $C = \Psi(C)$.

Proof. We begin by noticing that

$$\sigma_B \cdot C = (A - p \times (\delta_B \cdot C))$$

which gives $\Psi(C) = \texttt{OnlineDixon}(\sigma_B \cdot C, \sigma_B, C) = C$.

Next, we compute the shift of Ψ. Recall that Proposition 3.4 states that Algorithm `OnlineDixon` is on-line with respect to its first argument and has a shift 1 with respect to its third input. As a consequence, for any $n \in \mathbb{N}$, the computation of $\Psi(y)_n$ reads at most the p-adic coefficients $[A - p \times (\delta_B \cdot Y)]_i$ and Y_j with $0 \leqslant i \leqslant n$ and $0 \leqslant j \leqslant n-1$. Since $[A - p \times (\delta_B \cdot Y)]_i$ reads at most Y_k for $0 \leqslant k \leqslant n-1$, we have proved our assertion. □

The following proposition analyze the complexity in our two cases of interest. Let us recall that $\mathsf{R}(d)$ denotes the cost of the relaxed multiplication at precision N.

Proposition 3.6. *Let $B \in \mathcal{M}_{r\times r}(R_p)$ and $A \in \mathcal{M}_{r\times s}(R_p)$ be two p-adic matrices and note $d := \lambda(B)$ the length of B. Let $\alpha := \alpha(B)$ be the displacement rank of B and $d' := \max(\lambda(G), \lambda(H))$ where G, H are generators for B. We solve the linear system $B \cdot C = A$ at precision N so we can always assume that $N \geqslant d$.*

Then the computation costs of $C = B^{-1} \cdot A$ are displayed in the following table, depending on the matrix representation of B.

Dense representation	$\mathcal{O}\Big(r^\omega + N\,\frac{\mathsf{MMR}(r,s,d)}{d}\Big)$
Structured matrices	$\mathcal{O}\Big(\alpha^2\,\mathsf{M}(r)\log(r) + \alpha\,s\,N\,\frac{\mathsf{R}(r,d')}{d'}\Big)$

Table 3.7. *Cost of solving linear system for finite length matrices*

Proof. Proposition 3.5 tells us that for any shifted decomposition (σ_B, δ_B) of B, the s.l.p.

$$\Psi\colon \quad Y \mapsto \texttt{OnlineDixon}(A - p \times (\delta_B \cdot Y), \sigma_B, Y)$$

satisfies the hypothesis of Proposition 2.17, taking into consideration Remark 2.18. Therefore, this s.l.p. can be used to compute $B^{-1} \cdot A$ at precision N in the time necessary to evaluate Ψ at y.

If B is a dense matrix, we take the shifted decomposition $(\sigma_B, \delta_B) := (\sigma(B), \delta(B))$ of B. Since the p-adic matrix δ_B has length lesser or equal to d, the multiplication $\delta_B \cdot Y$ costs $\mathcal{O}(N\,\mathsf{MMR}(r,s,d)/d)$. Using Proposition 3.3 with $\lambda(B_0) = 1$, we conclude that the cost of solving for dense matrices is

$$\mathcal{O}(N\,\mathsf{MMR}(r,s,d)/d) + \mathcal{O}(\mathsf{MM}(r,s,1)\,N) + \mathcal{O}(\mathsf{MM}(r,r,1)).$$

For structured matrices B, we write B as

$$\begin{aligned} B &= \sum_{i=0}^{\alpha} L(G_i) \cdot U(H_i) \\ &= \underbrace{\sum_{i=0}^{\alpha} L(\sigma(G_i)) \cdot U(\sigma(H_i))}_{\sigma_B} + \underbrace{p\Big(\sum_{i=0}^{\alpha} L(\delta(G_i)) \cdot U(H_i) + L(\sigma(G_i)) \cdot U(\delta(H_i))\Big)}_{\delta_B}. \end{aligned}$$

We take the (σ_B, δ_B) of previous equation as a shifted decomposition for B. The important point is that $\alpha(\sigma_B) \leqslant \alpha$ and $\alpha(\delta_B) \leqslant 2\,\alpha$. Moreover the displacement generators of σ_B are $\sigma(G)$, $\sigma(H)$, which have length 1. The matrix multiplication $\delta_B \cdot Y$ costs $\mathcal{O}(\alpha\ N\ \mathsf{R}(r,\ d')/d')$. The cost of applying `OnlineDixon` is given by Proposition 3.3. Summing up, the cost of solving for structured matrices is

$$\mathcal{O}(\alpha\,s\,N\,\mathsf{R}(r,d')/d') + \mathcal{O}(\alpha\,s\,\mathsf{M}(r)\,N) + \mathcal{O}(\alpha^2\,\mathsf{M}(r)\log(r)). \qquad \square$$

Remark 3.7. For matrices B with length greater than 1, one should use Algorithm `OnlineSolve` instead of `OnlineDixon` as it is faster. However, we had to present Algorithm `OnlineDixon` for any matrices B of finite length. Indeed, in the latter proof in the structured matrix case, the matrix σ_B has length $\lceil \log_2(r) \rceil$.

3.4 Implementation and Timings

In this section, we display computation times in milliseconds for the univariate polynomial root lifting and for the computation of the product of the inverse of a matrix with a vector or with another square matrix. Timings are measured using one core of an INTEL XEON X5650 at 2.67 GHz running LINUX, GMP 5.0.2 [G+91] and setting $p = 536871001$ a 30 bit prime number.

Our implementation is available in the files whose names begin with `series_carry` or `p_adic` in the C++ library ALGEBRAMIX of MATHEMAGIX.

In the following tables, the first line, "Newton" corresponds to the classical Newton iteration [GG03, Algorithm 9.2] used in the zealous model. The second line "Relaxed" corresponds to our best variant. The last line gives a few details about which variant is used. We make use of the naive variant "N" and the relaxed variant "R". These variants differ only by the on-line multiplication algorithm used in Algorithm `OnlineEvaluationStep` inside Algorithm `OnlineRecursivePadic` to compute the recursive p-adics (see Section 2.2.2). The naive variant calls Algorithm `LazyMulStep` of Section 1.1.1.3, whereas the relaxed variant calls Algorithm `RelaxedProductStep` of Section 1.1.3.4. In fact, since we work on p-adic integers, the relaxed version uses an implementation of Algorithm $\texttt{Binary\_Mul\_Padic}_p$ from [BHL11, Section 3.2], which is a p-adic integer variant of Algorithm `RelaxedProductStep`.

Furthermore, when the precision is high, we make use of blocks of size 32 or 1024. That means, that at first, we compute the solution f up to precision 32 as $F_0 = f_0 + \cdots + f_{31}\, p^{31}$ with the variant "N". Then, we say that our solution can be seen as a p^{32}-adic integer $F = F_0 + \cdots + F_n p^{32n} + \cdots$ and the algorithm runs with F_0 as the initial condition. Then, each F_n is decomposed in base p to retrieve $f_{32n}, \ldots, f_{32n+31}$. Although it is competitive, the initialization of F can be quite expensive. "BN" means that F is computed with the variant "N", while "BR" means it is with the variant "R". Finally, if the precision is high enough, one may want to compute F with blocks of size 32, and therefore f with blocks of size 1024. "B^{2}N" (resp. "B^{2}R") means that f and F are computed up to precision 32 with the variant "N" and then, the p^{1024}-adic solution is computed with the variant "N" (resp. "R").

The next two tables correspond to timings for computing $B^{-1} \cdot A$ at precision n, with $A, B \in \mathcal{M}_{r\times r}(\mathbb{Z}_p)$. In this case, it is fair to compare our relaxed algorithm with Newton's iteration algorithm because they both have quasi-optimal cost $\tilde{\mathcal{O}}(r^\omega\, n)$. We see that "Relaxed" performs well.

n	4	16	64	2^8	2^{10}	2^{12}	2^{14}	2^{16}
Newton	0.097	0.22	0.89	6.8	59	490	3400	20000
Relaxed	0.15	0.61	3.1	8.1	38	335	1600	14000
Variant	N	N	N	BN	BN	BN	B^{2}N	B^{2}N

Table 3.8. Square matrices of size $r = 8$

n	4	16	64	2^8	2^{10}
Newton	930	2600	14000	140000	1300000
Relaxed	3600	18000	53000	150000	1000000
Variant	N	N	N	BN	BN

Table 3.9. Square matrices of size $r = 128$

Now, we solve integer linear systems and retrieve the solutions over $\mathbb{Q}$, using the rational number reconstruction [GG03, Section 5.10]. We set q as p to the power 2^j and pick at random a square matrix B of size r with coefficients in $M = \{0, \ldots, q-1\}$. We solve $B \cdot C = A$ with a random vector A. Because we deal with q-adic numbers at low precision, we only use the variant "N". We compared with LinBox [Lin08] and IML [CS04] but we do not display the timings of IML within LinBox because they are about 10 times slower. As LinBox and IML are designed for big matrices and small integers, it is not surprising that "Relaxed" performs better on these small matrices with big integers.

j	0	2	4	6	8	10	12
LinBox	1.0	1.4	3.6	25	310	4700	77000
Relaxed	0.10	0.24	0.58	2.1	14	110	760

Table 3.10. Integer linear system of size $r = 4$

j	0	2	4	6	8	10
LinBox	5.9	25	170	1900	27000	480000
Relaxed	24	150	360	2000	14000	90000

Table 3.11. Integer linear system of size $r = 32$

In fact, when $j \leqslant 3$, there is a major overhead coming from the use of Gmp. Indeed, in these cases, we transform q-adic numbers into p-adic numbers, compute up to the necessary precision and call the rational reconstruction.

Acknowledgments

We would like to thank J. van der Hoeven, M. Giusti, G. Lecerf, M. Mezzarobba and É. Schost for their helpful comments and remarks. For their help with LinBox, we thank B. Boyer and J.-G. Dumas.