Computing the N-th term of a q-holonomic sequence

Alin Bostan
Inria, Palaiseau, France

ABSTRACT
In 1977, Strassen invented a famous baby-step / giant-step algorithm that computes the factorial \( N! \) in arithmetic complexity quasi-linear in \( \sqrt{N} \). In 1988, the Chudnovsky brothers generalized Strassen’s algorithm to the computation of the N-th term of any holonomic sequence in the same arithmetic complexity. We design \( q \)-analogues of these algorithms. We first extend Strassen’s algorithm to the computation of the \( q \)-factorial of \( N \), then Chudnovskys’ algorithm to the computation of the N-th term of any \( q \)-holonomic sequence. Both algorithms work in arithmetic complexity quasi-linear in \( \sqrt{N} \).

We describe various algorithmic consequences, including the acceleration of polynomial and rational solving of linear \( q \)-differential equations, and the fast evaluation of large classes of polynomials including a family recently considered by Nogneng and Schost.

ACM Reference Format:

1 INTRODUCTION
A classical question in algebraic complexity theory is: how fast can one evaluate a univariate polynomial at one point? The precise formulation of this question depends on the model of computation. We will mainly focus on the arithmetic complexity model, in which one counts base field operations at unit cost.

Horner’s rule evaluates a polynomial \( P \) in \( O(\deg(P)) \) operations. Ostrowski [86] conjectured in 1954 that this is optimal for generic polynomials (i.e., whose coefficients are algebraically independent). This optimality result was proved a few years later by Pan [88].

However, most polynomials that one might wish to evaluate have coefficients which are not algebraically independent. Paterson and Stockmeyer [89] showed that for any field \( K \), an arbitrary polynomial \( P \in K[x] \) of degree \( n \) can be evaluated using \( O(\sqrt{n}) \) non-scalar multiplications; however, their algorithm uses a linear amount of scalar multiplications, so it is not well adapted to the evaluation at points from the base field \( K \), since in this case the total arithmetic complexity remains linear in \( n \).

On the other hand, for some families of polynomials, one can do much better. Typical examples are \( x^n \) and \( P_n(x) := x^{n-1} + \cdots + x + 1 \), which can be evaluated by repeated squaring in \( O(\log n) \) operations. (Note that for \( P_n(x) \) such a fast algorithm needs to perform division.) By contrast, a family \( P_n(x) \) of univariate polynomials is called hard to compute if the complexity of the evaluation of \( P_n \) grows at least like a power in \( \deg(P_n) \), whatever the algorithm used.

Paterson and Stockmeyer [89] proved the existence of polynomials in \( \mathbb{K}[x] \) which are hard to compute. Specific families of hard-to-compute polynomials were first exhibited by Strassen [104]. The techniques were refined and improved by Borodin and Cook [23], Lipton [80] and Schnorr [99], who produced explicit examples of polynomials whose evaluation requires a number of operations roughly linear in \( \sqrt{n} \). Subsequently, various methods have been developed to produce similar results on lower bounds, e.g., by Heintz and Sieveking [61] using algebraic geometry, and by Aldaz et al. [8] using a combinatorial approach. The topic is vast and very well summarized in the book by Bürgisser, Clausen and Shokrollahi [33].

In this article, we focus on upper bounds, that is on the design of fast algorithms for special families of polynomials, which are hard to compute, but easier to evaluate than generic polynomials. For instance, for the degree-\( \lceil \frac{n}{2} \rceil \)-polynomial \( Q_n(x) := P_1(x) \cdots P_n(x) \), a complexity in \( O(n) \) is clearly achievable. We will see in §2.1 that one can do better, and attain a cost which is almost linear in \( \sqrt{n} \) (up to logarithmic factors in \( n \)). Another example is \( R_n(x) := \sum_{k=0}^{n} x^{k^2} \), of degree \( n^2 \), and whose evaluation can also be performed in complexity quasi-linear in \( \sqrt{n} \), as shown recently by Nogneng and Schost [84] (see §2.2). In both cases, these complexities are obtained by clever although somehow ad-hoc algorithms. The starting point of our work was the question whether these algorithms for \( Q_n(x) \) and \( R_n(x) \) could be treated in an unified way, which would allow to evaluate other families of polynomials in a similar complexity.

The answer to this question turns out to be positive. The key idea, very simple and natural, is to view both examples as particular cases of the following general question: given a \( q \)-holonomic sequence, that is, a sequence satisfying a linear recurrence with polynomial coefficients in \( q \) and \( q^a \), how fast can one compute its N-th term?

In the more classical case of holonomic sequences (satisfying recurrences with polynomial coefficients in the index \( n \)), fast algorithms exist for the computation of the N-th term. They rely on a basic block, which is the computation of the factorial term \( N! \) in arithmetic complexity quasi-linear in \( \sqrt{N} \), using an algorithm due to Strassen [105]. The Chudnovsky brothers extended in [37] Strassen’s algorithm to the computation of the N-th term of any holonomic sequence in arithmetic complexity quasi-linear in \( \sqrt{N} \).

Our main contribution consists in transferring these results to the \( q \)-holonomic framework. It turns out that the resulting algorithms are actually simpler in the \( q \)-holonomic case than in the usual holonomic setting, essentially because multipoint evaluation on arithmetic progressions used as a subroutine in Strassen’s and Chudnovskys’ algorithms is replaced by multipoint evaluation on geometric progressions, which is considerably simpler [31].
A consequence of our results is that the following apparently unrelated polynomials / rational functions can be evaluated fast (note the change in notation, with variable x denoted now by q):

- \( A_n(q) \), the generating function of the number of partitions into \( n \) positive integers each occurring at most twice [111], i.e., the coefficient of \( x^n \) in the product \( \prod_{k \geq 1} (1 + q^k x + q^{2k} x^2) \)
- \( B_n(q) := \prod_{i=1}^{\infty} (1 - q^i) \mod q^n \), by Euler’s pentagonal theorem [87, §5], \( B_n(q) = 1 + \sum_{i=1}^{\infty} (-1)^i \left( \frac{q^{(3i+1)/2} + q^{3i+1}/2}{n} \right) \)

- The number \( C_n(q) \) of \( 2n \times 2n \) upper-triangular matrices over \( \mathbb{F}_q \) (the finite field with \( q \) elements), whose square is the zero matrix; \( C_n(q) \) is given by \([43, 71]\)

\[
C_n(q) = \sum_{j=0}^{\infty} \binom{2n}{n-3j} \left( \binom{2n}{n-3j-1} \right) q^{n^2+3j^2-j}.
\]

The common feature, exploited by the new algorithm, is that the three sequences \( A_n(q) \), \( B_n(q) \) and \( C_n(q) \) are \( q \)-holonomic. Actually, \( q \)-holonomic sequences are ubiquitous, so the range of application of our results is quite broad. This stems from the fact that \( q \)-holonomic sequences are coefficient sequences of power series satisfying \( q \)-difference equations, or equivalently, \( q \)-shift (or, \( q \)-difference) equations. From that perspective, our topic becomes intimately connected with \( q \)-calculus. The roots of \( q \)-calculus are in works of famous mathematicians such as Rothe [97], Gauss [55] and Heine [60]. The topic gained renewed interest in the first half of the 20th century, with the work, both on the formal and analytic aspects, of Tanner [106], Jackson [65–67], Carmichael [35], Mason [82], Adams [5, 6], Trjitzinsky [108], Le Caine [79] and Hahn [57], to name just a few. Modern accounts of the various aspects of the theory (including historical ones) can be found in [41, 44].

One of the reasons for interest in \( q \)-difference equations is that, formally, as \( q \) tends to 1, the \( q \)-derivative \( \frac{(qx)-f(x)}{(q-1)x} \) tends to \( f'(x) \), thus to every differential equation corresponds a \( q \)-difference equation which goes formally to the differential equation as \( q \to 1 \). In nice cases, (some of) the solutions of the \( q \)-difference equation go to solutions of the associated differential equation as \( q \to 1 \). An early example of such a good deformation behavior is given by the basic hypergeometric equation of Heine [60].

In computer algebra, \( q \)-holonomic sequences were considered starting from the early nineties, in the context of computer-generated proofs of identities in the seminal paper by Wilf and Zeilberger [110], notably in Section 5 (“Generalization to \( q \)-sums and \( q \)-multisums”) and in Section 6.4 (“\( q \)-sums and integrals”). Creative telescoping algorithms for (proper) \( q \)-hypergeometric sequences are discussed in various references [22, 36, 91]; several implementations of those algorithms are described for instance in [68, 73, 90, 95, 103]. Algorithms for computing polynomial, rational and \( q \)-hypergeometric solutions of \( q \)-differential equations were designed by Abramov and collaborators [1–4, 69]. These algorithms are important for several reasons. One is that they lie at the heart of the vast generalization by Chyzak [38, 39] of the Wilf and Zeilberger algorithmic theory, for the treatment of general \( q \)-holonomic (not only \( q \)-hypergeometric) symbolic summation and integration via creative telescoping. In that context, a multivariate notion of \( q \)-holonomy is needed; the foundations of the theory were laid by Zeilberger [114] and Sabbah [98] (in the language of D-modules), see also [36, § 2.5] and [52].

The simplest non-trivial \( q \)-holonomic sequence is \( n! \), which combinatorially counts the number of permutations of \( n \) objects. If instead of direct counting, one assigns to every permutation \( \pi \) its number of inversions \( inv(\pi) \), i.e., the number of pairs \( 1 \leq i < j \leq n \) with \( \pi(i) > \pi(j) \), the refined count (by size and number of inversions) is \([n]_q! := (1+q)(1+q^2)\cdots(1+q+\cdots+q^{n-1})\). This is the \( q \)-analogue of \( n! \), and it is the simplest non-trivial \( q \)-holonomic sequence.

There is also a natural \( q \)-analog of the binomial coefficients, called the Gaussian coefficients, defined by \( \binom{n}{k}_q := \frac{[n]_q!}{[k]_q![n-k]_q!} \). They have many counting interpretations, e.g., they count the \( k \)-dimensional subspaces of \( \mathbb{F}_q^n \) (points on Grassmannians over \( \mathbb{F}_q \)).

There are \( q \)-analogs to (almost) everything. To select just two basic examples, the \( q \)-analog of the binomial theorem is

\[
\prod_{k=1}^{n} (1 + q^{k-1} x) = \sum_{k=0}^{n} \binom{n}{k}_q q^{k} x^k.
\]

and the \( q \)-version of the Chu-Vandermonde identity is

\[
\sum_{k=0}^{n} q^k \binom{m}{k}_q \binom{n}{k}_q = \binom{m+n}{n}_q.
\]

The ubiquity of \( q \)-holonomic sequences is manifest in plenty of fields of mathematics: partition theory [11, 12, 81, 102, 111] and other subfields of combinatorics [13, 43, 45, 70, 71, 112]; theta functions and modular forms [18, 53, 77, 78, 113]; special functions [24, 64, 72] and in particular orthogonal polynomials [74, 75]; algebraic geometry [42], representation theory [63]; knot theory [48–52]; Galois theory [62]; number theory [40, 85].

The main message of this article is that for any example of \( q \)-holonomic sequence occurring in those various fields, one can compute selected coefficients faster than by a direct algorithm.

**Complexity basics.** We estimate the complexities of algorithms by counting arithmetic operations \((+,-,\times,\%\) in the base field \( \mathbb{K} \) at unit cost. We use standard complexity notation, such as \( M(d) \) for the cost of degree-\( d \) multiplication in \( \mathbb{K}[x] \) and \( \theta \) for feasible exponents of matrix multiplication. The best known upper bounds [46] are \( \theta < 2.3729 \). Most arithmetic operations on univariate polynomials of degree \( d \) in \( \mathbb{K}[x] \) can be performed in quasi-linear complexity \( O(d) \): multiplication, shift, interpolation, gcd, resultant, etc.\(^*\). A key feature of these results is the reduction to fast polynomial multiplication, which can be performed in time \( M(d) = O(d \log d \log \log d) \) [34, 101]. A general reference for these questions is [54].

---

\*As usual, the notation \( O(\cdot) \) is used to hide polylogarithmic factors in the argument.
2.1 De Feo’s question

Here is our first example, emerging from a question asked to the author by Luca De Feo\(^1\); this was the starting point of the article.

Let \( q \) be an element of the field \( \mathbb{K} \), and consider the polynomial \[ F(x) := \prod_{i=0}^{n-1} (x - q^i) \in \mathbb{K}[x]. \] (3)

Given another element \( a \in \mathbb{K} \), how fast can one evaluate \( F(a) \)?

Obviously, a direct algorithm consists in computing the successive powers \( q, q^2, \ldots, q^{n-1} \) using \( O(n) \) operations in \( \mathbb{K} \), then computing the elements \( a - q, a - q^2, \ldots, a - q^{n-1} \) using \( O(n) \) more operations in \( \mathbb{K} \), and finally returning their product. The total arithmetic cost of this algorithm is \( O(n) \), linear in the degree of \( F \).

Is it possible to do better? The answer is positive, as one can use the following baby step / giant step strategy, in which, in order to simplify things, we assume that \( n \) is a perfect square\(^5\), \( n = s^2 \).

Algorithm 1

(1) (Baby step) Compute the values of \( q, q^2, \ldots, q^{s-1} \), and deduce the coefficients of the polynomial \( G(x) := \prod_{j=0}^{s-1} (x - q^j) \).

(2) (Giant step) Compute \( Q := q^s, Q^2, \ldots, Q^{s-1} \), and deduce the coefficients of the polynomial \( H(x) := \prod_{k=0}^{n-1} (a - Q^k - x) \).

(3) Return the result \( \text{Res}(G, H) \).

By the basic property of resultants, the output of this algorithm is \[ \text{Res}(G, H) = \prod_{j=0}^{s-1} H(q^j) = \prod_{j=0}^{s-1} \left( (a - q^j) \right)^{\frac{n-1}{s}} = (a - q)^n = F(a). \]

Using the fast subproduct tree algorithm [54, Algorithm 10.3], one can perform the baby step (1) as well as the giant step (2) in \( O(M(\sqrt{n}) \log n) \) operations in \( \mathbb{K} \), and by [54, Corollary 11.19] the same cost can be achieved for the resultant computation in Step (3). Using fast polynomial multiplication, we conclude that \( F(a) \) can be computed in arithmetic complexity quasi-linear in \( \sqrt{n} \).

It is possible to speed up the previous algorithm by a logarithmic factor in \( n \) using a slightly different scheme, still based on a baby step / giant step strategy, but exploiting the fact that the roots of \( F \) are in geometric progression. Again, we assume that \( n = s^2 \) is a perfect square. This alternative algorithm goes as follows. Note that it is very close in spirit to Pollard’s algorithm [92, p. 523].

Algorithm 2

(1) (Baby step) Compute \( q, q^2, \ldots, q^{s-1} \), and deduce the coefficients of the polynomial \( P(x) := \prod_{j=0}^{s-1} (a - q^j - x) \).

(2) (Giant step) Compute \( Q := q^s, Q^2, \ldots, Q^{s-1} \), and evaluate \( P \) at \( 1, Q, Q^2, \ldots, Q^{s-1} \).

(3) Return the product \( P(1)P(Q) \cdots P(Q^{s-1}) \).

Obviously, the output of this algorithm is \[ \prod_{k=0}^{s-1} (a - q^k - q^k) = \prod_{k=0}^{s-1} (a - q^k) = F(a). \]

As pointed out in the Remarks after the proof of [31, Lemma 1], one can compute \( P(x) = P_1(x) = \prod_{j=0}^{n-1} (a - q^j \cdot x) \) in Step (1) without computing the subproduct tree, by using a divide-and-conquer scheme which exploits the fact that \( P_{2k}(x) = P_k(x) \cdot P_k(q^k x) \) and \( P_{2k+1}(x) = P_k(x) \cdot P_k(q^j x) \cdot (a - q^{k+1} x) \). The cost of this algorithm is \( O(M(\sqrt{n})) \) operations in \( \mathbb{K} \). As for Step (2), one can use the fast chirp transforms algorithms of Rabiner, Schafer and Rader [94] and of Bluestein [21]. These algorithms rely on the following observation: writing \( \alpha = Q^{ij} \cdot Q^{-j} \cdot Q^{-i} \) and \( P(x) = \sum_{j=0}^{s-1} c_j x^j \) implies that the needed values \( P(Q^j) = \sum_{i=0}^{s-1} c_j Q^i, 0 \leq j < s, \) are \[ P(Q^j) = Q^{-j} \sum_{j=0}^{s-1} c_j Q^{-i} \cdot Q^{ij}, 0 \leq j < s, \]

in which the sum is simply the coefficient of \( x^{s-1+j} \) in the product \[ \left( \sum_{k=0}^{s-1} c_k Q^{-k} \right) x^{s-1+j}. \]

This polynomial product can be computed in \( 2M(s) \) operations (and even in \( M(s) + O(s) \) using the transposition principle [30, 58], since only the median coefficients \( x^{s-1}, \ldots, x^{s-2} \) are actually needed). In conclusion, Step (2) can also be performed in \( O(M(\sqrt{n})) \) operations in \( \mathbb{K} \), and thus \( O(M(\sqrt{n})) \) is the total cost of this second algorithm.

We have chosen to detail this second algorithm for several reasons: not only because it is faster by a factor \( \log(n) \) compared to the first one, but more importantly because it has a simple structure, which will be generalizable to our general \( q \)-holonomic setting.

2.2 Evaluation of some sparse polynomials

Let us now consider the sequence of (sparse polynomial) sums

\[ v_N^{(n,a,b)}(q) = \sum_{n=0}^{N-1} p^n q^a n^b + b n, \]

where \( p \in \mathbb{K} \) and \( a, b \in \mathbb{Q} \) such that \( 2a, a + b \) are both integers. Typical examples are (truncated) modular forms, which are ubiquitous in complex analysis [18], number theory [113] and combinatorics [11]. For instance, the Jacobi theta function \( \vartheta \) depends on two complex variables \( z \in \mathbb{C} \), and \( \tau \in \mathbb{C} \) with \( \Im(\tau) > 0 \), and it is defined by

\[ \vartheta_3(z; \tau) = \sum_{n=-\infty}^{\infty} e^{\pi i (n^2 \tau + 2nz)} = 1 + 2 \sum_{n=1}^{\infty} q^{n^2}, \]

where \( q = e^{\pi i \tau} \) is the nome (\( |q| < 1 \)) and \( \eta = e^{2\pi i \tau} \). Here, \( \mathbb{K} = \mathbb{C} \). Another example is the Dedekind eta function, appearing in Euler’s famous pentagonal theorem [81], which has a similar form

\[ q^{\frac{1}{24}} \cdot \left( 1 + \sum_{n=1}^{\infty} (-1)^n \left( q^n + q^{n+1} \right) \right), \quad \text{with} \quad q = e^{2\pi i \tau}. \]

Moreover, sums of the form \( v_N^{(n,a,b)}(q) = \sum_{n=0}^{N-1} a^n q^b + b n \), over \( \mathbb{K} = \mathbb{Z} \) or \( \mathbb{K} = \mathbb{F}_2 \), crucially occur in a recent algorithm by Tao, Crott and Helfgott [107] for the efficient construction of prime numbers in given intervals, e.g., in the context of effective versions of Bertrand’s postulate. Actually, (the proof of) Lemma 3.1 in [107] contains the first sublinear complexity result for the evaluation of the sum \( v_N^{(n,a,b)}(q) \) at an arbitrary point \( q \); namely, the cost is \( O(N^{-\theta/3}) \), where \( \theta \in [2, 3] \) is any feasible exponent for matrix multiplication.

\(^1\)Private (email) communication, 10 January, 2020.

\(^2\)If \( n \) is not a perfect square, then one can compute \( F(a) \) as \( F(a) = F_1(a)F_2(a) \), where \( F_1(a) := \prod_{i=0}^{\sqrt{n} - 1} (a - q^i) \) is computed as in Algorithm 1, while \( F_2(a) := \prod_{i=\sqrt{n}}^{n-1} (a - q^i) \) can be computed naively, since \( n - \lfloor \sqrt{n} \rfloor^2 = O(\sqrt{n}). \)
As an immediate consequence, the sequence with general term \( v_N^{(p,a,b)}(q) = \sum_{j=0}^{N-1} b^j q^{ja^2+bi} = \sum_{k=0}^{r-1} \sum_{j=0}^{k-1} b^{j+k} q^{ja^2+bi} 
abla (j+k) \) can be written
\[
\sum_{k=0}^{r-1} \sum_{j=0}^{k-1} b^{j+k} q^{ja^2+bi} \cdot \varphi(q^{a^2k}), \quad \varphi(y) = \sum_{j=0}^{r-1} b^j q^{ja^2+bi} y^j.
\]

Therefore, the computation of \( v_N^{(p,a,b)}(q) \) can be reduced essentially to the multipoint evaluation of the degree-\( s \) polynomial \( P \) at \( s \) points (in geometric progression), with arithmetic cost \( \mathcal{O}(N^2) \).

We now describe an alternative algorithm, of same complexity \( \mathcal{O}(N^2) \), with a larger constant in the big-\( O \)-estimate, but whose advantage is its potential of generality.

Let us denote by \( u_n(q) \) the summand \( p^n q^{an^2+bn} \). Clearly, the sequence \( u_n(q) \) satisfies the recurrence relation
\[
u_{n+1}(q) = A(q, q^n) \cdot u_n(q), \quad \text{where} \quad A(x, y) = px^{a+b} y^2.
\]

As an immediate consequence, the sequence with general term \( u_n(q) = \sum_{k=0}^{n-1} b_k q^{ka^n+bk} \) satisfies a similar recurrence relation
\[
u_{n+1}(q) = A(q, q^n) \cdot (v_{n+1}(q) - \nu_n(q)),
\]
with initial conditions \( \nu_0(q) = 0 \) and \( \nu_1(q) = 1 \). This scalar recurrence of order two is equivalent to the first-order matrix recurrence
\[
\begin{bmatrix}
u_{n+2} \\
u_{n+1}
\end{bmatrix} = 
\begin{bmatrix}
A(q, q^n) + 1 & -A(q, q^n)
\end{bmatrix}
\begin{bmatrix}
u_{n+1} \\
u_n
\end{bmatrix}.
\]

By unrolling this matrix recurrence, we deduce that
\[
\begin{bmatrix}
u_{n+1} \\
u_n
\end{bmatrix} = M(q^{n-1}) \begin{bmatrix}
u_n \\
u_{n-1}
\end{bmatrix} = M(q^{n-1}) \cdots M(q) M(1) \begin{bmatrix}1 \\
0
\end{bmatrix},
\]
where
\[
M(x) = \begin{bmatrix} p^a x^2 + 1 & -p^a x^2 a^2 \\
0 & 1
\end{bmatrix}.
\]

Therefore, the computation of \( v_n(q) \) reduces to the computation of a matrix \( q \) factorial
\[
M(q^{N-1}) \cdots M(q) M(1)
\]
which can be performed by using a baby step / giant step strategy similar to the one of the second algorithm in \( \S 2.1 \). Again, we assume for simplicity that \( N = s^2 \) is a perfect square. The algorithm goes as follows.

Algorithm 3
(1) (Baby step) Compute \( q, q^2, \ldots, q^{s-1} \), then the coefficients of the matrix polynomial \( P(x) = M(q^{s-1}) \cdots M(q) M(x) \), using the baby step / giant step strategy similar to the one of the second algorithm in \( \S 2.1 \). Again, we assume for simplicity that \( N = s^2 \) is a perfect square. The algorithm goes as follows.

Algorithm 3
(1) (Baby step) Compute \( q, q^2, \ldots, q^{s-1} \), then the coefficients of the matrix polynomial \( P(x) = M(q^{s-1}) \cdots M(q) M(x) \), using the baby step / giant step strategy similar to the one of the second algorithm in \( \S 2.1 \). Again, we assume for simplicity that \( N = s^2 \) is a perfect square. The algorithm goes as follows.

Algorithm 3
(1) (Baby step) Compute \( q, q^2, \ldots, q^{s-1} \), then the coefficients of the matrix polynomial \( P(x) = M(q^{s-1}) \cdots M(q) M(x) \), using the baby step / giant step strategy similar to the one of the second algorithm in \( \S 2.1 \). Again, we assume for simplicity that \( N = s^2 \) is a perfect square. The algorithm goes as follows.

Algorithm 3
(1) (Baby step) Compute \( q, q^2, \ldots, q^{s-1} \), then the coefficients of the matrix polynomial \( P(x) = M(q^{s-1}) \cdots M(q) M(x) \), using the baby step / giant step strategy similar to the one of the second algorithm in \( \S 2.1 \). Again, we assume for simplicity that \( N = s^2 \) is a perfect square. The algorithm goes as follows.
As mentioned in the Introduction, q-holonomic sequences show up in various contexts. As an example, in (quantum) knot theory, the (\"colored\") Jones function of a (framed oriented) knot (in 3-space), is a powerful knot invariant, related to the Alexander polynomial [16] and it is a q-holonomic sequence of Laurent polynomials [51]. Its recurrence equations are themselves of interest as they closely related with the A-polynomial of a knot, via the AJ conjecture [47], verified in some cases using massive computer algebra calculations [49].

It is well known that the class of q-holonomic sequences is closed under several operations, such as addition, multiplication, twisting by roots of unity and substitution of q by q^r, with r \in \mathbb{Q} [48, 52, 103]. All these closure properties are effective, i.e., they can be executed algorithmically on the level of q-recurrences. Several computer algebra packages are available for the manipulation of q-holonomic sequences, e.g., in the Mathematica packages qGeneratingFunctions [68], HolonomicFunctions [76], and the Maple packages qsum [22], qFPS [103] and QDifferenceEquations.

A simple, but useful fact is that the order-r scalar recurrence (5) can be translated into a first-order recurrence on r \times r matrices:

\[
\begin{bmatrix}
-u_{n+r} \\
\vdots \\
-u_{n+1}
\end{bmatrix} = 
\begin{bmatrix}
\frac{-s_q}{c_q} & \cdots & \frac{-s_2}{c_2} & \frac{-s_1}{c_1} \\
\vdots & \ddots & \vdots & \vdots \\
0 & \cdots & 0 & 1
\end{bmatrix}
\begin{bmatrix}
u_{n+r-1} \\
\vdots \\
u_n
\end{bmatrix}.
\]

(8)

In particular, the N-th term of a q-holonomic sequence is simply expressible in terms of the matrix q-factorial

\[M(q^{N-1}) \cdots M(q)M(1),\]

where M(q^n) denotes the companion matrix from equation (8).

3.2 Computation of the q-factorial

The next result contains the promised q-analogue of Strassen’s result on the computation of N!.

\textbf{Theorem 3.2.} Let \( \mathbb{K} \) be a field of characteristic zero, let \( q \in \mathbb{K} \) and \( N \in \mathbb{N} \). The q-factorial \( [N]_q! \) can be computed using \( O(M(\sqrt{N})) \) operations in \( \mathbb{K} \). The same is true for the q-Pochhammer symbol \((a; q)_N\) for any \( a \in \mathbb{K} \).

\textbf{Proof.} If \( q = 0 \), then \([N]_q! = 1\); if \( q = 1 \) then \([N]_q! = N!\) can be computed in \( O(M(\sqrt{N})) \) operations using [29, Theorem 14]. We can assume that \( q \notin \{0, 1\} \). We have \([N]_q! = r^N \cdot (q)_N = r^N \cdot F(1)\), where \( r := (1 - q)^{-1} \) and \( F(x) := \prod_{i=1}^{N} (x - q^i) \). Algorithm 2 can be used to compute \((q)_N = F(1)\) in \( O(M(\sqrt{N})) \) operations in \( \mathbb{K} \). The cost of computing \( r^N \) is \( O(\log N) \) and it is negligible. The assertion concerning the q-Pochhammer symbol follows from the equality \((a; q)_N = F(a^{-1}) \cdot a^N\) for \( a \neq 0 \).

The assumption on the characteristic of \( \mathbb{K} \) can be relaxed, and replaced by appropriate invertibility assumptions, as in [29, §6].

\textbf{Corollary 3.3.} Under the assumptions of Theorem 3.2 and for any \( n \in \mathbb{N} \), one can compute in \( O(M(\sqrt{N})) \) operations in \( \mathbb{K} \):

- the q-binomial coefficient \( \binom{N}{n}_q \);
- the coefficient of \( x^n \) in the polynomial \( \prod_{k=1}^{N} (1 + q^{k-1}x) \);
- the element \( \frac{n!}{q^n} + \frac{(n-1)!}{q^{n-1}} + \cdots + \frac{1!}{q^1} + \frac{0!}{q^0} \).

3.3 \( N \)-th term of a q-holonomic sequence

We give the promised q-analogue of Chudnovsky’s result on the computation of the \( N \)-th term of an arbitrary holonomic sequence.

\textbf{Theorem 3.4.} Let \( \mathbb{K} \) be a field of characteristic zero, \( q \in \mathbb{K} \) and \( N \in \mathbb{N} \). Let \( (u_n(q))_{n \geq 0} \) be a q-holonomic sequence satisfying recurrence (5), and assume that \( c_r(q, q^k) \) is nonzero for \( k = 0, \ldots, N-1 \). Then, the term \( u_N(q) \) can be computed in \( O(M(\sqrt{N})) \) operations in \( \mathbb{K} \).

\textbf{Proof.} Using equation (8), it is enough to show that the matrix q-factorial \( M(q^{N-1}) \cdots M(q)M(1) \) can be computed in \( O(M(\sqrt{N})) \), where \( M(q^n) \) denotes the companion matrix from equation (8).

Algorithm 3 adapts \textit{mutatis mutandis} to this effect.

\textbf{Corollary 3.5.} Let \( \mathbb{K} \) be a field of characteristic zero, \( q \in \mathbb{K} \) and \( N \in \mathbb{N} \). Let \( e_q(x) \) be the q-exponential, defined by

\[ e_q(x) := \sum_{n \geq 0} \frac{x^n}{[n]!}_q \]

and let \( E_N^q(x) = e_q(x) \mod x^N \) be its truncation to precision \( N \). For any \( a \in \mathbb{K} \), one can compute \( E_N^q(a) \) in \( O(M(\sqrt{N})) \) operations in \( \mathbb{K} \).

\textbf{Proof.} Denote the summand \( \frac{a^n}{[n]!}_q \) by \( u_n(q) \). Then \( (u_n(q))_n \) is q-hypergeometric, and satisfies the recurrence \( [n + 1]q_{n+1} - au_n = 0 \), therefore \( v_N(q) := \sum_{n=0}^{N} u_n(q) \) satisfies the second-order recurrence \( [n + 1]q(v_{n+1}(q) - v_n(q)) = a(v_{n+1}(q) - v_n(q)) \). Applying Theorem 3.4 to \( v_N(q) \) concludes the proof.

Remark that the same result holds if \( e_q(x) \) is replaced by any power series satisfying a q-differential equation. For instance, one can evaluate fast all truncations of Heine’s q-hypergeometric series

\[ 2\Phi_1([a, b], [c]; q, x) := \sum_{n \geq 0} \frac{\binom{a}{n} \binom{b}{n} \binom{c}{n}}{\binom{d}{n}} \cdot \frac{x^n}{(q^n)} \]
computing the summands $q^n$ one after the other, before summing them, has bit complexity $\bigO(N^3 \cdot B)$. This is not (quasi-)optimal with respect to the output size. Can one do better? The answer is “yes”. It is sufficient to use the $q$-holonomic character of $u_N(q)$, and to reduce its computation to that of a $q$-factorial matrix (9) as in §2.2. Now the point is that, instead of using baby steps / giant steps, it is a better idea to use binary splitting. The complexity of this approach becomes then quasi-optimal, that is $\bigO(N^2 \cdot B)$, which is quasi-linear in the bitsize of the output. The following general result is proved in a similar way.

**Theorem 3.7.** Under the assumptions of Theorem 3.4, with $\mathbb{K} = \mathbb{Q}$, the term $u_N(q)$ can be computed in $\bigO(N^2 \cdot B)$ bit operations, where $B$ is the bitsize of $q$.

As a corollary, (truncated) solutions of $q$-differential equations can be evaluated using the same (quasi-)linear bit complexity. This result should be viewed as the $q$-analogue of the classical fact that holonomic functions can be evaluated fast using binary splitting, a 1988 result by the Chudnovsky brothers [37, §6], anticipated a decade earlier (without proof) by Schroeppe1 and Salamin in item 178 of [17]; see [19, §12] for a good survey on binary splitting.

4 APPLICATIONS

4.1 Combinatorial $q$-holonomic sequences

As already mentioned, many $q$-holonomic sequences arise in combinatorics, for example in connection with the enumeration of lattice polygons, where $q$-analogs of the Catalan numbers $\frac{1}{n+1} \binom{2n}{n}$ occur naturally [13, 45, 56, 93], or in the enumeration of special families of matrices with coefficients in $\mathbb{F}_q$ [43, 70, 71, 112], where sequences related to the Gaussian coefficients $\binom{n}{k}_q$ also show up.

A huge subfield of combinatorics is the theory of partitions [11], where $q$-holonomic sequences occur as early as in the famous Roger-Ramanujan identities [96], such as the identity

$$1 + \sum_{n \geq 1} \frac{q^n}{(1 - q) \cdots (1 - q^n)} = \prod_{n \geq 0} \frac{1}{(1 - q^{n+1}) (1 - q^{2n+4})},$$

which translates the fact that the number of partitions of $n$ into parts that differ by at least two is equal to the number of partitions of $n$ into parts congruent to 1 or 4 modulo 5. Andrews and his collaborators [9, 10, 14] laid the foundations of a theory able to capture the $q$-holonomy of any generating function of a so-called linked partition ideal, see e.g., Theorem 4.1 in [10, §4] and [11, Chapter 8].

As a consequence, a virtually infinite number of special families of polynomials coming from partitions can be evaluated fast. For instance, a basic example is the family of truncated polynomials

$$F_n(x) := \prod_{k=1}^{\infty} (1 - x^k) \mod x^n,$$

which can be evaluated fast due to our results and to the identity

$$F_N(q) = \sum_{\binom{n+1}{2} \leq N} (-1)^{n} (2n + 1)q^{\binom{n+1}{2}}.$$

4.2 Evaluation of $q$-orthogonal polynomials

In the theory of special functions, orthogonal polynomials play an important role. There exists an extension to the $q$-framework of the theory, see e.g., Chapter 9 in Ernst’s book [44]. Amongst the most basic examples, the discrete $q$-Hermite polynomials [7, 15] are defined by their $q$-exponential generating function

$$\sum_{n \geq 0} F_n(x) \frac{t^n}{[n]_q!} = \frac{e_q(xt)}{e_q(t)[e_q(x) - t]},$$

and therefore they satisfy the second-order linear recurrence

$$F_{n+1,q}(x) = xF_n(x) - (1 - q^n)q^{n-1}F_{n-1,q}(x), \quad n \geq 1,$$

with initial conditions $F_{0,q}(x) = 1, F_{1,q}(x) = x$. From there, it follows that for any $a \in \mathbb{K}$, the sequence $(F_{n,q}(a))_{n \geq 0}$ is $q$-holonomic, thus the evaluation of the $n$-th polynomial at $x = a$ can be computed fast. The same is true for the continuous $q$-Hermite polynomials, for which $2aH_n,q(a) = H_{n+1,q}(a) + (1 - q^n)H_{n-1,q}(a)$ for $n \geq 1$, and $H_0,q(a) = 1, H_{1,q}(a) = 2a$. More generally, our results in §3 imply that any family of $q$-orthogonal polynomials can be evaluated fast.

4.3 Polynomial and rational solutions of $q$-differential equations

The computation of polynomial and rational solutions of linear differential equations lies at the heart of several important algorithms, for computing hypergeometric and Liouvillian solutions, for factoring and for computing differential Galois groups [109]. Creative telescoping algorithms (of second generation) for multiple integration with parameters [39, 76] also rely on computing rational solutions, or deciding their existence. The situation is completely similar for $q$-differential equations: improving algorithms for polynomial and rational solutions of such equations is important in finding $q$-hypergeometric solutions [4], in computing $q$-differential Galois groups [62], and in performing $q$-creative telescoping [39, 75, 76].

In both differential and $q$-differential cases, algorithms for computing polynomial solutions proceed in two distinct phases: (i) compute a degree bound $N$, potentially exponentially large in the equation size; (ii) reduce the problem of computing polynomial solutions of degree at most $N$ to linear algebra. Abramov, Bronstein and Petkovšek showed in [1] that, in step (ii), linear algebra in size $N$ can be replaced by solving a much smaller system, of polynomial size. However, setting up this smaller system still requires linear time in $N$, essentially by unrolling a $(q)$-linear recurrence up to terms of indices close to $N$. For differential (and difference) equations, this step has been improved in [27, 28], by using the Chudnovsky’s algorithms for computing fast the $N$-th term of a holonomic sequence. This allows for instance to decide (non-)existence of polynomial solutions in sublinear time $\bigO(\sqrt{N})$. Moreover, when polynomial solutions exist, one can represent / manipulate them in compact form using the recurrence and initial terms as a compact data structure.

The same improvements can be transferred to $q$-differential equations, in order to improve the existing algorithms [1–3, 69]. In this case, setting up the smaller system in phase (ii) amounts to computing fast the $N$-th term of a $q$ holonomic sequence, and this can be done using our results in §3.

A technical subtlety is that, as pointed out in [1, §4.3], it is not obvious in the $q$-differential case how to guarantee the non-singularity of the $q$-recurrence on the coefficients of the solution. This induces potential technical complications similar to the ones for polynomial solutions of differential equations in small characteristic, which can nevertheless be overcome by adapting the approach described in [12, §3.2].
\section*{4.4 $q$-hypergeometric creative telescoping}

In the case of differential and difference hypergeometric creative telescoping, it was demonstrated in \cite{Abramov97} that the compact representation for polynomial solutions can be used as an efficient data structure, and can be applied to speed-up the computation of Gosper forms and Zeilberger’s classical summation algorithm \cite{Zeilberger90}. The key to these improvements lies in the fast computation of the $N$-th term of a holonomic sequence, together with the close relation between Gosper’s algorithm and the algorithms for rational solutions. Similarly, in the $q$-differential case, Koornwinder’s $q$-Gosper algorithm \cite{Koornwinder92, Koornwinder95} is closely connected to Abramov’s algorithm for computing rational solutions \cite[\S 2]{Al-SalamCarlitz34}, and this makes it possible to transfer the improvements for rational solutions to the $q$-Gosper algorithm. This leads in turn improvements upon Koornwinder’s algorithm for $q$-hypergeometric summation \cite{Koornwinder95}, along the same lines as in the differential and difference case \cite{Abramov97}.

\section{5 CONCLUSION AND FUTURE WORK}

We have shown that selected terms of $q$-holonomic sequences can be computed fast, the key being the extension of classical algorithms by Strassen and the Chudnovsky brothers, in the holonomic case. We have also demonstrated through many examples that this basic algorithmic improvement has many other algorithmic implications, notably on the faster evaluation of many families of polynomials and the acceleration of algorithms for $q$-differential equations.

Here are some interesting questions that we would like to investigate in the future.

1. \textbf{(Computing curvatures of $q$-differential equations)} In the differential case, $p$-curvatures can be computed fast \cite{Bluestein72, Bluestein76}. What about the $q$-differential analogue? One strong motivation comes from the fact that the $q$-analogue \cite{AndrewsAskey92} of Grothendieck’s conjecture (relating equations over $Q$ with their reductions modulo primes $p$) is proved \cite{Zeilberger90}. This could be used to improve the computation of rational solutions.

2. \textbf{(Computing points on $q$-curves)} Counting efficiently points on (hyper)-elliptic curves leads to questions like: for $a, b \in \mathbb{Z}$, compute the coeff. of $x^p$ in $G_p(x) := (x^2 + ax + b)^{\frac{p-1}{2}}$ modulo $p$, for one \cite{Bezivin82} or several \cite{Bezivin84} primes $p$. A natural extension is to ask the same with $G_p(x)$ replaced by $\prod_{k=1}^{n} (q^{kx^k} + ax + b)$. This might have applications related to 1., or to counting points on $q$-deformations \cite{Borodin97}.

3. \textbf{(Computing $q$-deformed real numbers)} Recently, Morier-Genoud and Ovsienko \cite{MorierGenoudOvsienko12} introduced $q$-analogues of real numbers. How fast can one compute (truncations / evaluations of) quantized versions of numbers like $e$ or $\pi$?

4. \textbf{(Evaluating more polynomials)} Is it possible to evaluate fast polynomials of the form $\sum_{n=0}^{N} x^n$, for $n \geq 3$, and many others which escape the $q$-holonomic class?

\textbf{Acknowledgements.} I would like to thank Luca De Feo for his initial question, who motivated this work, and for the very interesting subsequent discussions. This work was supported in part by DeRerumNatura ANR-19-CE40-0018.


[59] E. Heine. Untersuchungen über die Reihe $(1-\alpha q^{-1}z)^{-\frac{1}{2}} + \left(1-\alpha q^{-1}z\right)^{-\frac{1}{2}} + \left(1-\alpha q^{-1}z\right)^{-\frac{1}{2}} + \cdots$. J. reine angew. Math., 34:285–328, 1847.


