Online order basis algorithm and its impact on block Wiedemann algorithm

Pascal Giorgi
LIRMM, UM2 - CNRS
pascal.giorgi@lirmm.fr

Romain Lebreton
University of Waterloo
rlebreton@uwaterloo.ca

ABSTRACT

Order bases are a fundamental tool for linear algebra with polynomial coefficients. In particular, block Wiedemann methods are nowadays able to tackle large sparse matrix problems because they benefit from fast order basis algorithms. However, such fast algorithms suffer from two practical drawbacks: they are not designed for early termination and often require more knowledge on the input than necessary. In this paper, we propose an online algorithm for order basis which allows for both early termination and minimal input requirement while keeping quasi-optimal complexity in the order. Using this algorithm inside block Wiedemann methods leads to an improvement of their practical performances by a constant factor.

1. INTRODUCTION

Order bases (also called sigma bases) are the cornerstone of efficient algorithms with polynomial matrices over finite fields. Their use in algorithms for determinant, column reduction [6] or minimal nullspace basis [21] has reduced these problems to the simpler computation of polynomial matrix multiplication. A complete panel of these reductions can be found in [11, 19].

Since their introduction in 1994 [1], order basis computations are quasi-optimal in the order of approximation. Fast linear algebra has been included later on for square matrices [6], then for any matrix dimensions [20].

Historically, order basis were introduced in [1] to give a generalization of Hermite-Padé approximants in the matrix case. It is nowadays well known that a minimal generator of a linearly generated scalar sequence is related to a specific Padé approximant. As a non trivial transposition to the matrix case, it has been shown in [18, 16] that order bases provide a fast solution to the problem of computing a minimal generator of a linearly generated matrix sequence. Note that many other approaches solve this problem, *e.g.* Toeplitz/Hankel solver, matrix Berlekamp-Massey or matrix extended Euclidean, but only few of them provide fast polynomial arithmetic together with fast matrix multiplication. We refer to [14] for a classification of all possible algorithms with their corresponding reference and complexity.

Modern sparse linear algebra over a finite field extensively relies on Coppersmith's block Wiedemann algorithm [4], which in turn computes a minimal matrix generating polynomial. In practical applications (e.g. integer factorization or rank of cohomology group), we do not always dispose *a priori* of a tight bound on the degree of the minimal generator. Early termination techniques are an important tool to deal with this issue but, to the best of our knowledge, no fast block Wiedemann algorithms enable early termination strategies in the order basis computation.

Hence we focus in this paper on the application of order basis to the block Wiedemann framework and we investigate the possibility to reduce its complexity in the context of early termination. In particular, existing fast order basis algorithms may compute superfluous coefficients of their power series matrix input when stopping early. This is the case of the fast algorithm PM-Basis of [6]. The cost of computing this power series matrix is dominant in block Wiedemann, and so it is of great interest to minimize the number of required coefficients.

Main results. We contribute by giving two new algorithms iPM-Basis and oPM-Basis. Algorithm iPM-Basis is a variant of PM-Basis suited to early termination. It consists of an iterative version of the recursive algorithm PM-Basis. This enables efficient early termination strategies in block Wiedemann methods. However, this is not optimal yet because we may have to pay the cost of computing more coefficients of the power series matrix than necessary.

Therefore we propose a fast online order basis oPM-Basis. Since *online* algorithms require minimal knowledge on the input by definition (see Section 5), we use oPM-Basis inside block Wiedemann algorithm and show that it has a positive impact both in theory (see Proposition 12) and practice (see Figure 4) in the context of early termination.

2. USEFUL MATERIALS

Let $\mathbb{K}$ be a field. If $A = \sum_i A_i x^i$ is in $\mathbb{K}[x]^{m\times n}$ or $\mathbb{K}[[x]]^{m\times n}$, and i, j are integers with $i \leqslant j$, then we write

$$A_{i\dots j} = A_i + A_{i+1}x + \cdots + A_{j-1}x^{j-i-1},$$

so that $A_{i\dots j}$ has degree less than $j - i$.

Let $A \in \mathbb{K}[x]^{m\times m}$ and $B \in \mathbb{K}[x]^{m\times n}$ with A of degree less than d. Then, the middle product $\mathrm{MP}(A, B, d, h)$ of A and B is defined as the part $D_{d-1\dots h}$ of the product $D := A \cdot B$,

so that $\deg(\mathrm{MP}(A,B,d,h)) < h-d+1$. The *balanced* case corresponds to $h = 2d-1$; we denote this $\mathrm{MP}(A,B,d)$.

2.1 Shifted degree

The notion of shifted degree of a polynomial matrix is an extension of the classic polynomial degree where some scalings are introduced either by row or by column. In particular, the shifted $\vec{s}$-row degree of a row vector $v \in \mathbb{K}[x]^{1\times n}$ with $\vec{s} = [s_1, \ldots, s_n]$ corresponds to $\max_{1\le i\le n}(s_i + \deg v[i]) \in \mathbb{N}$.

This naturally extends to any matrix $A \in \mathbb{K}[x]^{m\times n}$ by considering the $\vec{s}$-row degrees of the rows of A. The following lemma states the compatibility of the $\vec{s}$-row degree with matrix multiplication.

LEMMA 1 ([19, LEMMA 2.4]). *If the $\vec{u}$-row degree of $B \in \mathbb{K}[x]^{k\times n}$ is bounded by the vector $\vec{v} \in \mathbb{Z}^k$ and the $\vec{v}$-row degree of $A \in \mathbb{K}[x]^{m\times k}$ is bounded by $\vec{w} \in \mathbb{Z}^m$ then the $\vec{u}$-row degree of AB is bounded by $\vec{w}$.*

2.2 Row reduced matrix

A matrix $R \in \mathbb{K}[x]^{m\times m}$ is said to be $\vec{s}$-row reduced if the $\vec{s}$-row degree of R is lesser or equal to the $\vec{s}$-row degree of UR for any unimodular matrix U. Note that we say that a $\vec{s}$-row degree $\vec{u}$ is greater or equal to $\vec{v}$ if all its components are greater or equal. Despite this is not a total order, it always exists a matrix with minimal $\vec{s}$-row degree in the set of UR for U unimodular [19, Corollary 1.21].

The following lemma gives a criteria to certify that a matrix is $\vec{s}$-row reduced. Denote by $x^{\vec{s}}$ the $n\times n$ diagonal matrix with entries $[x^{s_i}, \ldots, x^{s_n}]$.

LEMMA 2 ([2, 19]). *Let $\vec{u}$ be the $\vec{s}$-row degree of a matrix $R \in \mathbb{K}[x]^{m\times m}$. Then R is $\vec{s}$-row reduced if and only if the leading coefficient matrix $T = x^{-\vec{u}}Rx^{\vec{s}} \bmod x$ of R for the $\vec{s}$ degree has full rank.*

This row reduced form is not unique as is. Extra conditions on R are required for unicity; this is known as the Popov form [2]. The following lemma establishes a condition on the product of reduced matrices to be reduced.

LEMMA 3 ([19, LEMMA 2.18]). *If B is a full-rank $\vec{u}$-row reduced matrix with its $\vec{u}$-row degree bounded by $\vec{v}$, then AB is $\vec{u}$-row reduced if and only if A is $\vec{v}$-row reduced.*

3. ORDER BASIS ALGORITHMS

Let $F = \sum_{i\geqslant 0} F_i x^i \in \mathbb{K}[[x]]^{m\times n}$ be a matrix of power series with $m \ge n$, σ a positive integer and (F,σ) be the $\mathbb{K}[x]$-module of $v \in \mathbb{K}[x]^{1\times m}$ such that $vF \equiv 0 \bmod x^\sigma$. A polynomial matrix P is a *(left) order basis* of F of order $\sigma \in \mathbb{N}^*$ and shift $\vec{s} \in \mathbb{N}^m$ if the rows of P form a basis of (F,σ), P is non-singular and P is $\vec{s}$-row reduced (see [20, 19] for more details).

Without loss of generality, we only consider in this paper the case $n = O(m)$ with a balanced shift $\vec{s}$ as in [6]. Indeed the techniques of [20] allow to reduce the general situation to our particular case.

Nowadays there are mainly two algorithms for computing order bases. In this section, we propose a new presentation of these two algorithms of [6] using the modern framework of order basis from [19].

Throughout the paper, the order basis algorithms, except M-Basis, will take as input a power series matrix $F \in \mathbb{K}[[x]]^{m\times n}$, a shift vector $\vec{s} \in \mathbb{N}^m$ and an approximation order $\sigma \in \mathbb{N}^*$, and return a $(\sigma,\vec{s})$ order basis P of F and its $\vec{s}$-row degree $\vec{u}$.

3.1 The base case (order 1)

In order to simplify the presentation of all following order basis algorithms, we treat separately the case $\sigma = 1$. This case corresponds to finding a polynomial matrix M of minimal $\vec{s}$-row degree such that $MF \equiv 0 \bmod x$. This problem roughly reduces to the computation of the column reduced echelon form of the kernel of $\pi F \in \mathbb{K}^{m\times n}$, where π is a permutation matrix such that the sequence $\pi\vec{s}$ is increasing. Hence, one can use the PLE decomposition [10] over $\mathbb{K}$ for this matter. In this decomposition, P is a permutation matrix, L is a lower unit triangular matrix and E is the echelon form.

Algorithm 1: Basis$(F,\vec{s})$

Input: $F \in \mathbb{K}^{m\times n}$, $\vec{s} \in \mathbb{N}^m$
Output: $M \in \mathbb{K}[x]^{m\times m}$, $\vec{u} \in \mathbb{N}^m$

1: Find the permutation π s.t. $\pi\vec{s}$ is sorted increasingly
2: Compute the PLE decomposition τLE of πF
3: Let $r = \mathtt{rank}(E)$ and write $L = \begin{bmatrix} L_r & \\ G & I_{m-r}\end{bmatrix}$
4: $M := \begin{bmatrix} xI_r & \\ -GL_r^{-1} & I_{m-r}\end{bmatrix}\tau^{-1}\pi$
5: $\vec{u} := \tau^{-1}\pi\vec{s} + \begin{bmatrix}1_r & 0_{n-r}\end{bmatrix}^T$
6: **return** $M, \vec{u}$

PROPOSITION 4. *Algorithm Basis outputs a $(1,\vec{s})$ order basis M of F and its $\vec{s}$-row degree $\vec{u}$. Its complexity is $O(mnr^{\omega-2})$ operations in $\mathbb{K}$, where r corresponds to the rank of F.*

PROOF. For the correctness of the algorithm, we need to prove that $MF \equiv 0 \bmod x$, $\det M \neq 0$ and M is $\vec{s}$-row reduced. By definition of the row echelon form, it is clear that the $m-r$ last rows of E are zero. Consequently the last $m-r$ rows of MF are also zero. Since the first r rows of M equal to xI_r, it is clear that $MF \equiv 0 \bmod x$. Then $\det M = \pm x^r \neq 0$. Finally we assume, without loss of generality, that the permutations π and τ are equal to the identity. This assumption implies that $\vec{s}$ is already sorted increasingly and that $\vec{u} = [s_1+1, \ldots, s_r+1, s_{r+1}, \ldots, s_m]$. Therefore, the matrix $T = x^{-\vec{u}}Mx^{\vec{s}} = I_m + O(x^{-1})_{x\to\infty}$, which implies by Lemma 2 that M is $\vec{s}$-row reduced and then concludes the proof of correctness of algorithm Basis. The complexity of Basis algorithm is dominated by the cost $O(mnr^{\omega-2})$ of PLE decomposition [10]. □

3.2 Quadratic algorithm

To compute a $(\sigma,\vec{s})$-order basis, Algorithm M-Basis presented in [6] works iteratively by induction on the approximation order.

PROPOSITION 5. *Algorithm M-Basis is correct and its complexity is $O(m^\omega\sigma^2)$ operations in $\mathbb{K}$.*

PROOF. Let us prove the correctness of Algorithm M-Basis by induction on the order σ. The case $\sigma = 1$ is already proven as it corresponds to Algorithm Basis. Assume P_{k-1}

Algorithm 2: M-Basis$(F, \sigma, \vec{s})$

Input: $F \in \mathbb{K}[[x]]^{m\times n}$, $\vec{s} \in \mathbb{N}^m$ and $\sigma \in \mathbb{N}^*$
Output: $P \in \mathbb{K}[x]^{m\times m}$ and $\vec{u} \in \mathbb{N}^m$

1: $P_0, \vec{u}_0 := \mathsf{Basis}(F \bmod x, \vec{s})$
2: **for** $k = 1$ **to** $\sigma - 1$ **do**
3: $\quad F' := (x^{-k} P_{k-1} F) \bmod x$
4: $\quad M_k, \vec{u}_k := \mathsf{Basis}(F',\ \vec{u}_{k-1})$
5: $\quad P_k := M_k P_{k-1}$
6: **return** $P_{\sigma-1}, \vec{u}_{\sigma-1}$

is a $(k, \vec{s})$-order basis of F, and let us prove that $M_k P_{k-1}$ is a $(k+1, \vec{s})$-order basis. By definition, $F' \in \mathbb{K}^{m\times n}$ is the first non-zero term of $P_{k-1}F$. Since $M_k F' \equiv 0 \bmod x$, $P_k = M_k P_{k-1}$ is a basis for $(F, k+1)$. One need to show that P_k is $\vec{s}$-row reduced to finish the proof. By definition of order basis, P_{k-1} is $\vec{s}$-row reduced and its $\vec{s}$-row degree is bounded by $\vec{u}_{k-1}$. Similarly, M_k is $\vec{u}_{k-1}$-row reduced and its $\vec{u}_{k-1}$-row degree is bounded by $\vec{u}_k$. So it follows directly from Lemmas 1 and 3 that P_k is $\vec{s}$-row reduced and its $\vec{s}$-row degree is bounded by $\vec{u}_k$.

The complexity of M-Basis is dominated by step 5 which computes at most $k+1$ products of $m \times m$ matrices. Hence, the cost of each loop is bounded by $O(m^\omega k)$. Incorporating the latter cost in the for loop gives the announced complexity for M-Basis. □

Remark 1. Let θ_k be the maximum degree of P_k in M-Basis. In Proposition 5, we use $\theta_k = k$. However, for generic matrices and homogeneous shift $\vec{s}$, one can show that $\theta_k = \lceil kn/m \rceil$. As a consequence, the arithmetic complexity of M-Basis becomes $O(m^\omega \sigma \theta_\sigma) = O(m^{\omega-1} n \sigma^2)$.

3.3 Quasi-linear algorithm

A divide-and-conquer approach has been presented in [6], namely the PM-Basis algorithm. The idea is similar to the ones used in [15, 1]: it reduces the computation of one order basis of order σ to two order bases of order $\sigma/2$.

Algorithm 3: PM-Basis$(F,\ \sigma,\ \vec{s})$

Input: $F \in \mathbb{K}[[x]]^{m\times n}$, $\vec{s} \in \mathbb{N}^m$ and $\sigma \in \mathbb{N}^*$
Output: $P \in \mathbb{K}[x]^{m\times m}$ and $\vec{u} \in \mathbb{N}^m$

1: **if** $\sigma = 1$ **then**
2: $\quad$ **return** $\mathsf{Basis}(F \bmod x, \vec{s})$
3: **else**
4: $\quad P_l, \vec{u}_l := \mathsf{PM\text{-}Basis}(F,\ \lfloor \sigma/2 \rfloor, \vec{s})$
5: $\quad F' := \mathrm{MP}(P_l, F, \lfloor \sigma/2 \rfloor + 1, \sigma)$
6: $\quad P_h, \vec{u}_h := \mathsf{PM\text{-}Basis}(F', \lceil \sigma/2 \rceil, \vec{u}_l)$
7: **return** $P_h \cdot P_l, \vec{u}_h$

As explained in [6], this strategy allows to reduce the arithmetic complexity to $O(m^\omega \mathsf{M}(\sigma)\log(\sigma))$, where M denotes the arithmetic complexity of polynomial multiplication. Note that the proof of correctness of PM-Basis is a direct implication of Lemmas 1 and 3. Contrary to M-Basis, the recursive structure of PM-Basis makes it difficult to enable early termination.

4. FAST ITERATIVE ORDER BASIS

In this section, we give a variant of PM-Basis more suited to early termination. We present an original iterative variant iPM-Basis of the recursive algorithm PM-Basis. Let us denote $\nu_2(k)$ the valuation in 2 of any integer k. Let $k = \sum_{i=1}^{r} 2^{n_i}$ be the binary writing of k with $\nu_2(k) = n_1 < \cdots < n_r$. For $0 \leqslant j \leqslant r$, we define $\varphi(k, j) = \sum_{i=r-j+1}^{r} 2^{n_i}$ as the sum of the jth highest bits of k. For example, if $k = 7$ then $r = 3$ and $[\varphi(k,i)]_{0\leqslant i\leqslant 3} = [0, 4, 6, 7]$.

Algorithm 4: iPM-Basis'$(F, \sigma, \vec{s})$

Input: $F \in \mathbb{K}[[x]]^{m\times n}$, $\vec{s} \in \mathbb{N}^m$ and $\sigma \in \mathbb{N}^*$
Output: $P \in \mathbb{K}[x]^{m\times m}$ and $\vec{u} \in \mathbb{N}^m$

1: $F^{(0)} = F$
2: $M_0, \vec{u}_0 := \mathsf{Basis}(F \bmod x, \vec{s})$
3: **for** $k = 1$ **to** $\sigma - 1$ **do**
4: $\quad v := \nu_2(k)$
5: $\quad M^{(k)} = ((M_{k-1} \cdot M^{(k-2^0)}) \cdot M^{(k-2^1)}) \cdots M^{(k-2^{v-1})}$
6: $\quad F^{(k)} = \mathrm{MP}(M^{(k)}, F^{(k-2^v)}, 2^v + 1, 2^{v+1})$
7: $\quad M_k, \vec{u}_k := \mathsf{Basis}(F^{(k)} \bmod x,\ \vec{u}_{k-1})$
8: $v := \nu_2(\sigma)$, $\sigma = \sum_{i=1}^{r} 2^{n_i}$
9: $M^{(\sigma)} = ((M_{\sigma-1} \cdot M^{(\sigma-2^0)}) \cdot M^{(\sigma-2^1)}) \cdots M^{(\sigma-2^{v-1})}$
10: **return** $(M^{(\varphi(\sigma,r))} \cdot M^{(\varphi(\sigma,r-1))}) \cdots M^{(\varphi(\sigma,1))}, \vec{u}_{\sigma-1}$

It is easy to add a stop criteria to iPM-Basis' in the for loop after line 7. When σ is a power of two, Algorithms PM-Basis and iPM-Basis perform the same ordered sequence of polynomial matrix multiplication and call to Basis. Note that these algorithms differ in terms of memory management.

PROPOSITION 6. *For any $F, \vec{s}$ and $k \in \mathbb{N}$, Algorithms PM-Basis and iPM-Basis' have the same output and perform the same computations on the input $F, 2^k, \vec{s}$.*

PROOF. We proceed by induction. When $k = 0$, both algorithms perform only a call to Basis $(F \bmod x, \vec{s})$.

Now recursively, suppose the result true for $k < t$ and let us prove it for $k = t$. The program $\mathsf{PM}_t := \mathsf{PM\text{-}Basis}(F, 2^t, \vec{s})$ is made of 4 instructions, and we claim that they are mapped to subparts of $\mathsf{It}_t := \mathsf{iPM\text{-}Basis'}(F, 2^t, \vec{s})$ as follows:

1. line 4 of PM_t maps to the code $\mathsf{It}_{t,\mathrm{beg}}$ of It_t from the beginning until line 5 for $k = 2^{t-1}$;
2. line 5 of PM_t corresponds to line 6, $k = 2^{t-1}$ of It_t;
3. line 6 and 7 of PM_t are mapped to $\mathsf{It}_{t,\mathrm{end}}$ defined by the code from line 7, $k = 2^{t-1}$ to the end of It_t.

Our proposition follows from this former correspondence, which we now prove. Note that since σ is a power of two, iPM-Basis' outputs $M^{(\sigma)}$.

1. First the call to PM_{t-1} on line 4 of PM_t corresponds to It_{t-1} by induction hypothesis. The latter code and It_t share the same code until line 7, $k = 2^{t-1} - 1$. Then It_{t-1} outputs the product $M^{(2^{t-1})}$ of line 9, which corresponds to the product of It_t, line 5 for $k = 2^{t-1}$.

2. Let us prove that the inputs of both middle products are the same. By induction hypothesis, the outputs of PM_{t-1} and It_{t-1} are the same, that is $M^{(2^{t-1})}$ as we have seen before. Concerning the second input, note that $F^{(k-2^v)} = F^{(0)} = F$ for $k = 2^{t-1}$. So the middle products coincide and their outputs F' and $F^{(2^{t-1})}$ correspond.

3. By recurrence hypothesis, line 6 of PM_t corresponds to $\mathsf{It}'_{t-1} := \mathsf{iPM\text{-}Basis'}(F^{(2^{t-1})}, 2^{t-1}, \vec{u}_{2^{t-1}-1})$. Thus, we will

prove the correspondence of It'_{t-1} followed by line 7 of PM_t with $\mathsf{It}_{t,\mathrm{end}}$. Let us denote with an additional apostrophe the local variables $M_k, M^{(k)}$ and $F^{(k)}$ of It'_{t-1}. Let us prove by recurrence that for all $0 < k < 2^{t-1}$, $\vec{u}'_k = \vec{u}_{k+2^{t-1}}$, $M'_k = M_{k+2^{t-1}}$, $M'^{(k)} = M^{(k+2^{t-1})}$ and $F'^{(k)} = F^{(k+2^{t-1})}$.

The initialization of It'_{t-1} gives $F'^{(0)} = F^{(2^{t-1})}$ and $M'_0 = M_{2^{t-1}}$. Then recursively on $k < 2^{t-1}$; suppose the claim verified for indices lesser than k, and let us prove it for k. First $M'^{(k)} = M^{(k+2^{t-1})}$. Indeed both formulas of line 5 of iPM-Basis' match since $M'_{k-1} = M_{k-1+2^{t-1}}$, $\nu_2(k) = \nu_2(k + 2^{t-1})$ and $M'^{(k-2^i)} = M^{(k+2^{t-1}-2^i)}$ for all $0 \leqslant i < \nu_2(k)$. Then the middle product of line 6 have the same input and therefore $F'^{(k)} = F^{(k+2^{t-1})}$. Finally, $M'_k = M_{k+2^{t-1}}$ and $\vec{u}'_k = \vec{u}_{k+2^{t-1}}$ as they are the output of Basis on the same inputs.

At last, the products of line 9 of It'_{t-1} and $\mathsf{It}_{t,\mathrm{end}}$ differ only by one term, which corresponds to line 7 of PM_t. □

COROLLARY 7. *Algorithm iPM-Basis' outputs a $(\sigma, \vec{s})$ order basis P of F and its $\vec{s}$-row degree $\vec{u}$ in $O(m^\omega \mathsf{M}(\sigma)\log(\sigma))$ operations in $\mathbb{K}$.*

PROOF. The complexity of our algorithm follows from the one of Algorithm PM-Basis for orders σ that are powers of two. Concerning the correctness, note that $M^{(k)} = M_{k-1} \cdots M_{k-2^v}$ where $v = \nu_2(k)$. As a consequence, the output of Algorithm iPM-Basis' is $M_{\sigma-1} \cdots M_0$. The variables M_k of iPM-Basis' coincide with those of M-Basis because PM-Basis performs the same calls to Basis as M-Basis. Subsequently, the output of iPM-Basis' and M-Basis coincide and our output is a $(\sigma, \vec{s})$ order basis of F. □

We now propose an improved version iPM-Basis of Algorithm iPM-Basis' with respect to memory utilization. In iPM-Basis', we have $O(\sigma)$ variables $M^{(k)}$ and $F^{(k)}$. We can reduce this number to $O(\log(\sigma))$ by keeping only one variable $\bar{M}^{(v)}$ for all $M^{(k)}$ with the same valuation $v = \nu_2(k)$, the same for $F^{(k)}$ and one variable M for the last M_k.

Algorithm 5: iPM-Basis$(F, \sigma, \vec{s})$

Input: $F \in \mathbb{K}[[x]]^{m\times n}$, $\vec{s} \in \mathbb{N}^m$ and $\sigma \in \mathbb{N}^*$
Output: $P \in \mathbb{K}[x]^{m\times m}$ and $\vec{u} \in \mathbb{N}^m$

1: $\bar{F}^{(\infty)} = F$
2: $M, \vec{u}_0 := \mathsf{Basis}(F \bmod x, \vec{s})$
3: **for** $k = 1$ **to** $\sigma - 1$ **do**
4: $\quad v := \nu_2(k)$, $v' := \nu_2(k - 2^v)$
5: $\quad \bar{M}^{(v)} = ((M \cdot \bar{M}^{(0)}) \cdot \bar{M}^{(1)}) \cdots \bar{M}^{(v-1)}$
6: $\quad \bar{F}^{(v)} = \mathrm{MP}(\bar{M}^{(v)}, \bar{F}^{(v')}, 2^v + 1, 2^{v+1})$
7: $\quad M, \vec{u}_k := \mathsf{Basis}(\bar{F}^{(v)} \bmod x,\ \vec{u}_{k-1})$
8: $v := \nu_2(\sigma)$, $\sigma = \sum_{i=1}^r 2^{n_i}$
9: $\bar{M}^{(v)} = ((M \cdot \bar{M}^{(0)}) \cdot \bar{M}^{(1)}) \cdots \bar{M}^{(v-1)}$
10: **return** $(\bar{M}^{(n_1)} \cdot \bar{M}^{(n_2)}) \cdots \bar{M}^{(n_r)}, \vec{u}_{\sigma-1}$

This way, we store at each step k the $M^{(r)}$ and $F^{(r)}$ for indices r in the following set :

$$\mathcal{P}_k := \{r \leqslant k \text{ s.t. } r = \max\{s \leqslant k \mid \nu_2(s) = \nu_2(r)\}\}.$$

It remains to prove that the indices $(k - 2^i)_{1\leqslant i<v}$, $k - 2^v$ and $(\varphi(k, i))_{1\leqslant i\leqslant r}$ of respectively lines 5, 6 and 10 of iPM-Basis' belong to $\mathcal{P}_k$. For any $0 \leqslant i < \nu_2(k)$, let us prove that $k - 2^i \in \mathcal{P}_k$. First $\nu_2(k - 2^i) = i$. The only other candidate greater than $k - 2^i$ with valuation i would be k, but since $\nu_2(k) \neq i$, it must be that $k - 2^i \in \mathcal{P}_k$. Then $\varphi(k, i) \in \mathcal{P}_k$ for $1 \leqslant i \leqslant r$; the valuation of $\varphi(k, i)$ is greater than n_{r-i+1} and the next integer with such a valuation is greater than k. Finally $k - 2^v = \varphi(k, r - 1)$.

5. ONLINE ORDER BASIS ALGORITHM

The $(\sigma, \vec{s})$-order basis $M_{\sigma-1} \cdots M_0$ of F is a function of $F \bmod x^\sigma$; this can be seen easily on Algorithm M-Basis. However fast Algorithms PM-Basis and iPM-Basis may require more coefficients of F to compute the same order basis. The speed of these algorithms is attained by performing computations in advance, which in turn require more knowledge on F. More precisely, for any order σ such that $2^{t-1} < \sigma \leqslant 2^t$, Algorithm iPM-Basis reads $F \bmod x^{2^t}$ (see line 6 for $k = 2^{t-1}$).

In some applications, such as the block Wiedemann method, the cost of computing the input F is dominant and minimizing its required precision can have a practical impact. This will be the subject of Section 6.

The purpose of this section is to give an algorithm that requires minimal knowledge on F while keeping a quasi-optimal complexity in the order σ. Note that Algorithm M-Basis satisfies the first condition but not the second, and that PM-Basis (and iPM-Basis) is in the opposite situation.

5.1 Online algorithms

Let A be an algorithm with one of its input $i = [i_0, \ldots, i_n]$ and its output $o = [o_0, \ldots, o_n]$ that are sequences. We say that A is an *online* algorithm with respect to its input i if it reads at most $i_0, \ldots, i_j$ when computing o_j for any $0 \leqslant j \leqslant n$. This notion, first defined by [7], was popularized in the domain of Computer Algebra in [8] under the name of *relaxed* algorithms.

In our context of order basis algorithms, we consider the input F as the sequence of its power series coefficients F_i and the output to be the sequence $[M_0, \ldots, M_{\sigma-1}]$. To put it differently, we say that an order basis algorithm is *online* if it requires minimal knowledge on F, that is if it reads at most $F_0, \ldots, F_k$ when computing M_k. Our first example of online order basis algorithm is M-Basis; the computation of M_k requires $(x^{-k} P_{k-1} F) \bmod x$, which in turn depends only on the power series coefficients $F_0, \ldots, F_k$ of F.

Our objective in this section is to obtain a fast online order basis algorithm. In algorithm iPM-Basis, we have noticed that the middle product of step 5 may read more entries of F than necessary. Therefore, we need to perform this step differently.

5.2 Shifted online middle product

Let $A \in \mathbb{K}[x]^{m\times m}$ and $B \in \mathbb{K}[x]^{m\times n}$ with A of degree less than d. We are interested in computing the middle product $C = \mathrm{MP}(A, B, d, h)$ under the following constraint: we cannot use the polynomial coefficients A_{i+d} or B_{i+d} before we have computed the coefficients $C_0, \ldots, C_i$ of C. Compared to online algorithms, there is a shift in the indices of A and B due to the fact that the ith coefficient of $\mathrm{MP}(A, B, d, h)$ is the $(i + d - 1)$-th coefficient of $A \cdot B$. We will call such an algorithm a *shifted online* algorithm for the middle product.

Note that for our problem, the reading constraint affects only B since $A_{i+d} = 0$ by definition. Therefore we are only interested in algorithms for the middle product that

are shifted online with respect to B. Algorithms that are (shifted) online with respect to only one input out of two are called *(shifted) half-line* algorithms.

Let us focus on the balanced base $h = 2d-1$ for the rest of this section. As in the *offline* (not online) case, we have a naive algorithm for the middle product which consists in computing the full product $A_{0\dots 2d-1} \cdot B_{0\dots d}$ using an half-line algorithm w.r.t. B.

We denote by $\mathsf{H}(n)$ the arithmetic complexity of multiplying two such power series at precision n by a half-line algorithm. As with classic multiplication cost functions, we suppose that H is a super-linear, *i.e.* that $r\mathsf{H}(s) \leqslant \mathsf{H}(rs)$ for any integers r, s. The cost of the naive shifted online middle product algorithm is therefore $\mathsf{SMP}_{\mathsf{Naive}} := \mathsf{H}(2n-1)$.

We now propose a dedicated shifted online algorithm sMiddleProduct for the middle product that will save asymptotically a factor 2 in the complexity compared to the naive algorithm.

Algorithm 6: sMiddleProduct(A, B, d)

Input: $A \in \mathbb{K}[x]^{m\times m}$ of degree $< d$, $B \in \mathbb{K}[x]^{m\times n}$
Output: $\mathrm{MP}(A, B, d) \in \mathbb{K}[x]^{m\times n}$

1: **for** $k = 0$ **to** $d-1$ **do**
2: **if** $k = 0$ **then**
3: $C = (A_{0\dots d} \cdot B_{0\dots d})_{d-1\dots 2d-1} \in \mathbb{K}[x]^{m\times n}$
4: **else**
5: $m := \nu_2(k)$
6: $C = C + \mathrm{MP}(A_{0\dots 2^{m+1}-1}, B_{d+k-2^m\dots d+k}, 2^m)\ x^k$
7: **return** C

The scheme of computation of our algorithm is given in Figure 1 when $d = 8$. Decompose $C = \mathrm{MP}(A, B, d)$ into a lower and an upper part $C = L + xU$ where $L = (A_{0\dots d} \cdot B_{0\dots d}) \operatorname{div} x^{d-1}$ and $U = (A_{0\dots d-1} \cdot B_{d\dots 2d-1}) \bmod x^{d-1}$. At the first step of our algorithm, we are allowed to use $A_{0\dots d}$ and $B_{0\dots d}$ in order to compute $C_0 = L_0$. We chose to compute the whole L *via* a classic offline multiplication algorithm at cost $\mathsf{M}(d)$.

Let $B' := B_{d\dots 2d-1}$. For the computation of U, the reading constraint translates to the following: the computation of U_i must read at most the coefficients B'_i of B'. So we can use a half-line multiplication algorithm for this task, at cost $\mathsf{H}(d-1)$.

Altogether, Algorithm sMiddleProduct computes C in time $\mathsf{SMP}_{\mathsf{GL}} := \mathsf{M}(d) + \mathsf{H}(d-1)$. Since $\mathsf{M}(d) = o(\mathsf{H}(d))$ and $2\mathsf{H}(d-1) \leqslant \mathsf{H}(2d-1)$, we deduce that

$$2\,\mathsf{SMP}_{\mathsf{GL}}(d) \leqslant (1+o(1))\,\mathsf{SMP}_{\mathsf{Naive}}(d).$$

In Figure 1 and Algorithm sMiddleProduct (see lines 5-6), we chose to use the half-line multiplication of [9].

It turns out that the middle product of line 6 of iPM-Basis is unbalanced and does not exactly fit the settings of this section. However, the same ideas apply; we can cut the middle product in two parts, apply an offline multiplication on the lowest part and an half-line algorithm on the highest part.

5.3 Online order basis

For the purpose of a fast online order basis algorithm, we will use the lazy data structure to encode our polynomial matrices as in [8, 3]. In this data structure, polynomials are given as a promise, and the evaluation of its coefficients is delayed.

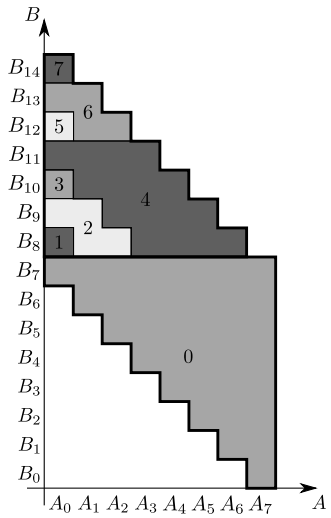

Figure 1: Shifted online middle product

We assume that we dispose of a class $O_{\mathsf{sMiddleProduct}}$ that perform the shifted online middle product using the lazy data structure. In this setting, step k of Algorithm sMiddleProduct is performed when the k-th coefficient of the middle product is asked for.

Algorithm 7: oPM-Basis$(F, \sigma, \vec{s})$

Input: $F \in \mathbb{K}[[x]]^{m\times n}$, $\vec{s} \in \mathbb{N}^m$ and $\sigma \in \mathbb{N}^*$
Output: $P \in \mathbb{K}[x]^{m\times m}$ and $\vec{u} \in \mathbb{N}^m$

1: $\bar{F}^{(\infty)} = F$
2: $M, \vec{u}_0 := \mathsf{Basis}(F \bmod x, \vec{s})$
3: **for** $k = 1$ **to** $\sigma - 1$ **do**
4: $v := \nu_2(k)$, $v' := \nu_2(k - 2^v)$
5: $\bar{M}^{(v)} = ((M \cdot \bar{M}^{(0)}) \cdot \bar{M}^{(1)}) \cdots \bar{M}^{(v-1)}$
6: $\bar{F}^{(v)} = O_{\mathsf{sMiddleProduct}}(\bar{M}^{(v)}, \bar{F}^{(v')}, 2^v + 1, 2^{v+1})$
7: $M, \vec{u}_k := \mathsf{Basis}(\bar{F}^{(v)} \bmod x,\ \vec{u}_{k-1})$
8: $v := \nu_2(\sigma)$, $\sigma = \sum_{i=1}^{r} 2^{n_i}$
9: $\bar{M}^{(v)} = ((M \cdot \bar{M}^{(0)}) \cdot \bar{M}^{(1)}) \cdots \bar{M}^{(v-1)}$
10: **return** $(\bar{M}^{(n_1)} \cdot \bar{M}^{(n_2)}) \cdots \bar{M}^{(n_r)}, \vec{u}_{\sigma-1}$

Proposition 8. *oPM-Basis is a correct online order basis algorithm of arithmetic complexity* $O(m^\omega \mathsf{M}(\sigma)\log^2(\sigma))$.

Proof. Remember that $\bar{F}^{(v)}$ of line 8 corresponds to $F^{(k)}$, which in turn equals to $(M_{k-1}\cdots M_0 F)_{k\dots k+2^v}$. Therefore, the call to $\bar{F}^{(v)} \bmod x$ on line 7 will read at most the entries $F_0, \dots, F_k$ of F at step k.

The cost analysis is similar to the one of PM-Basis, except that our middle product at order σ now costs $\mathsf{H}(\sigma) = O(\mathsf{M}(\sigma)\log(\sigma))$ instead of $\mathsf{M}(\sigma)$ [8]. Thus, Algorithm oPM-Basis costs $O(m^\omega \mathsf{H}(\sigma)\log(\sigma)) = O(m^\omega \mathsf{M}(\sigma)\log^2(\sigma))$, which is quasi-linear in σ. □

6. APPLICATION TO BLOCK WIEDEMANN

Let $A \in \mathcal{M}_N(\mathbb{K})$ be a sparse matrix, which means that it has $O(N\log(N))$ non-zero entries. Block Wiedemann methods are useful for the computation of the rank [17], the determinant [13] of such matrices and also for sparse linear systems solving [4]. The main ingredient of block Wiedemann algorithms is the computation of Π_U^{AV} the minimal matrix generating polynomial of the matrix sequence $S = (S_i)_{i=0}^{\sigma}$ such that $\Pi_U^{AV} \in \mathbb{K}[x]^{m\times m}$ and $S_i = (UA^iV) \in \mathbb{K}^{m\times n}$ where $m \geq n$. Here $U \in \mathbb{K}^{m\times N}$, $V \in \mathbb{K}^{N\times n}$ are chosen randomly. Note that A can be preconditioned to ensure structural properties.

Definition 9. The matrix $\Pi = \sum_{i=0}^{d} \Pi_i x^i \in \mathbb{K}[x]^{m\times m}$ is a *left matrix generating polynomial* of S if

$$\sum_{i=0}^{d} \Pi_i S_{i+j} = 0^{m\times n}, \quad \forall j \geq 0,$$

and $\det(\Pi) \neq 0$. It is said to be *minimal* if Π is row reduced.

As described in [18, 16], one can obtain Π_U^{AV} from a $(\sigma, \vec{s})$ order basis P of the matrix series $F = [\sum_{i=0}^{\sigma} S_i^T x^i \mid I_m]^T \in \mathbb{K}[[x]]^{2m\times n}$ where $\vec{s} = [0_m, 1_m]$. It is well known since Coppersmith work [4] that an order $\sigma_{\mathsf{C}} = \lceil N/m\rceil + \lceil N/n\rceil + O(1)$ is sufficient in most cases. However, this bound may be loose for certain class of matrices.

Theorem 2.12 of [13] gives a tighter bound by looking at the invariant factors of the characteristic matrix $(xI_N - A)$. Let μ be the sum of the degrees of the n largest invariant factors of $(xI_N - A)$. Then the minimal matrix generating polynomial of S can be derived from an order basis at precision $\sigma_{\mathsf{KV}} = \lceil \mu/m\rceil + \lceil \mu/n\rceil + O(1)$ [13, 17]. Note that μ gives a bound on the determinantal degree of Π_U^{AV}.

When the values of μ and σ_{KV} are not known, early termination is a good strategy to stop the course of the order basis algorithm. We use an heuristic method to detect that we have reached order σ_{KV}. For example, we can test if the m smallest values of the $\vec{s}$-row degree of P remain unchanged for a few consecutive orders. This latter condition is similar to getting consecutive zero discrepancies in the Berlekamp-Massey algorithm [12].

We will from now on set ourselves in the context of early termination in the block Wiedemann algorithm. So we have three choices for an order basis algorithms in block Wiedemann that allows early termination : the two online algorithms M-Basis and oPM-Basis, and the offline algorithm iPM-Basis.

We demonstrate below that online algorithms M-Basis and oPM-Basis can improve offline block Wiedemann complexity by a constant factor, up to 2, under some conditions. Whereas this gain can only be observed with M-Basis when the ratio μ/N is small, we show that this always the case with oPM-Basis whenever N is big. For the sake of simplicity we now consider that $m = n$ and m, σ are powers of two.

Computing S has dominant cost. Let δ be the minimal value such that a $(\delta, \vec{s})$-order basis of F contains Π_U^{AV}. We know that $2\mu/m < \delta \leq \sigma_{\mathsf{KV}} \leq \sigma_{\mathsf{C}}$ [13]. We will assume that our heuristic stop the order basis algorithm at order δ.

We now compare the complexity of the first two steps of the block Wiedemann algorithm, 1) computing the sequence of UA^iV's and 2) computing the $(\delta, \vec{s})$-order basis, depending on the choice of order basis algorithm (M-Basis, iPM-Basis or oPM-Basis). Respectively, we note these complexities $\mathcal{C}_{\text{M-Basis}}$, $\mathcal{C}_{\text{PM-Basis}}$ and $\mathcal{C}_{\text{oPM-Basis}}$.

M-Basis and oPM-Basis are both online algorithms that only require the first δ coefficients of F. On the contrary, iPM-Basis requires the first 2^t coefficients of F when $2^{t-1} < \delta \leq 2^t$. Let $\mathcal{X}$ be the cost of computing one coefficient of the sequence F, *i.e.* $UA^iV = U(A(A^{i-1}V))$. We assume without of loss generality that $\mathcal{X} = \Theta(m^2N)$, which is the case whenever $m = O(\log(N))$. Then

$$\mathcal{C}_{\text{M-Basis}} = \delta\mathcal{X} + O(m^\omega\delta^2) \quad (1)$$

$$\mathcal{C}_{\text{PM-Basis}} = 2^t\mathcal{X} + O(m^\omega\delta\log(\delta)) \quad (2)$$

$$\mathcal{C}_{\text{oPM-Basis}} = \delta\mathcal{X} + O(m^\omega\delta\log^2(\delta)) \quad (3)$$

The following two propositions establish order relation between these complexities.

PROPOSITION 10. *The ratio $\mathcal{C}_{\text{PM-Basis}}/\mathcal{C}_{\text{M-Basis}}$ tends to $2^t/\delta$ when μ/N tends to 0.*

PROOF. First, one has $\frac{\mathcal{C}_{\text{PM-Basis}}}{\mathcal{C}_{\text{M-Basis}}} = \frac{2^t/\delta+\varepsilon_1}{1+\varepsilon_2}$ where $\varepsilon_1 := m^\omega\log(\delta)/\mathcal{X}$ and $\varepsilon_2 := m^\omega\delta/\mathcal{X}$. Since $\varepsilon_1 = O(\varepsilon_2)$, it is enough to prove that ε_2 tends to 0 to imply that $\mathcal{C}_{\text{PM-Basis}}/\mathcal{C}_{\text{M-Basis}}$ tends to $2^t/\delta$ when μ/N tends to 0. Using $\mathcal{X} = \Theta(m^2N)$ and $\delta = O(\mu/m)$, we get that $\varepsilon_2 = O(m^{\omega-3}\mu/N)$ which tends to 0 as claimed. □

PROPOSITION 11. *The ratio $\mathcal{C}_{\text{PM-Basis}}/\mathcal{C}_{\text{oPM-Basis}}$ tends to $2^t/\delta$ when N tends to infinity.*

PROOF. The proof is similar but with $\varepsilon_2 := m^\omega\log^2(\delta)/\mathcal{X}$. From $\varepsilon_2 = O(m^{\omega-2}\log^2(\delta)/N)$ and using $\delta = O(N)$ and $m = O(\log(N))$, we deduce that $\varepsilon_2 = O(\log^3(N)/N)$ which tends to 0 when N tends to infinity. □

Analysis of the gain due to online algorithms. We will now assume that either we are using M-Basis and μ/N is small, or we are using oPM-Basis and N is large. Therefore Propositions 10 and 11 state that we gain a constant factor $K_\delta = 2^t/\delta$ compared to "offline" block Wiedemann. This constant K depends on δ and satisfies $1 \leqslant K < 2$. The upper bound is tight as it is easy to find δ such that K is as close as you want to 2. The following proposition states that the average gain is greater than 1.

PROPOSITION 12. *Let δ be a random variable uniformly distributed between 1 and $\sigma_{\mathsf{C}} = 2^r$. Then the expected value of K_δ tends to $2\ln 2 \simeq 1.39$ when r tends to infinity.*

PROOF. If δ were a random variable uniformly distributed between $2^{t-1}+1$ and 2^t, then its mean value would be

$$K^{(t)} := \frac{1}{2^t}\sum_{\delta=2^{t-1}+1}^{2^t} K_\delta = \sum_{\delta=2^{t-1}+1}^{2^t} \frac{1}{\delta} \xrightarrow[t\to\infty]{} \ln 2.$$

The actual mean value of K_δ is

$$\frac{1}{2^r}(K_1 + \sum_{i=1}^{r} 2^i K^{(i)}) = \frac{K_1}{2^r} + \sum_{j=0}^{r-1} 2^{-j} K^{(r-j)} \xrightarrow[r\to\infty]{} 2\ln 2. \quad \square$$

7. PRACTICAL PERFORMANCES

In this section, we discuss the implementation of our algorithms. The code is developed and distributed in the LinBox library (`www.linalg.org`). All the following benchmarks have been done on an Intel Core i7-4960HQ @ 2.6GHZ with 16Gb RAM and 4 cores. Only the computations of the block Wiedemann sequences have been done in parallel as it is the primary goal of this approach.

7.1 Polynomial matrix multiplication

Our fast implementation of polynomial matrix multiplication relies on several optimizations. First, we use the `FFLAS` package, available through the LinBox library, as a primitive for matrix multiplication. This allows to reach the processor's peak performance for matrix multiplication over word size prime field [5]. Secondly, we provide cache friendly DFT transforms for polynomials which uses SIMD vectorization, *i.e.* SSE4 instructions. As both matrices and polynomials must be stored contiguously to be cache friendly, we therefore implement two data structures, *i.e.* matrix of polynomials and polynomial of matrices, and efficient conversions in between.

m, d	MMX	FLINT 2.4	our code
$16, 1024$	**0.02s**	$0.26s$	$0.03s$
$16, 2048$	**0.04s**	$0.70s$	$0.06s$
$16, 4096$	**0.09s**	$1.68s$	$0.13s$
$16, 8192$	**0.20s**	$4.52s$	$0.28s$
$128, 512$	$1.00s$	$26.21s$	**0.82s**
$256, 256$	$4.00s$	$36.71s$	**1.75s**
$512, 512$	$69.19s$	$465.66s$	**19.64s**
$1024, 64$	$71.36s$	$115.52s$	**13.95s**
$2048, 32$	$298.27s$	$263.88s$	**48.90s**

Table 1: Computation times of polynomial matrix multiplication in $\mathbb{F}_p[x]^{m\times m}$ with degree d and p a 23-bit FFT prime.

Table 1 gives a quick preview of the performances of our code. For comparison purposes, we provide the times of two standard libraries: FLINT (`www.flintlib.org`) and Mathemagix (`www.mathemagix.org`) abbreviated as MMX. Our code provides similar or better performances than these reference libraries. In particular, our optimizations on DFT are closed to the ones of MMX, known to be efficient. On the other side, for large matrices, our code performs even better thanks to the use of optimized BLAS libraries which rely on the recent AVX2 instructions set of our processor. Note that MMX does not yet provide such a support in their `matrix_naive` type. This emphasises the benefit of the `FFLAS` approach which always makes use of the most recent technology. Regarding FLINT's performances, we explain the gap in performances by our use of cache friendly storage and vectorization.

7.2 Online middle product

Our fast shifted online middle product relies on the the half-line product of [9]. Our choice is driven by its complexity, roughly $\frac{1}{4}\mathsf{M}(\sigma)\log(\sigma)$, and its simplicity of implementation requiring only an efficient offline middle product. Therefore, we readily benefit from the optimizations of our polynomial matrix multiplication implementation.

Figure 2 illustrates the ratio of timings of both offline MiddleProduct and shifted online sMiddleProduct middle product algorithms with respect to polynomial matrix multiplication. As expected, the offline middle product behaves exactly as polynomial matrix multiplication and the shifted online ratio behaves like $1/4\log(\sigma)$.

7.3 Online order basis

As described in sections 3 and 5, fast order basis algorithms reduce to polynomial matrix multiplication. Our implementations of PM-Basis and oPM-Basis follow this reduction and therefore mainly use the middle products of 7.2 and the polynomial matrix multiplication of 7.1. In practice, for small value of σ, it appears that using fast order basis algorithm is not relevant and switching to quadratic approach as M-Basis improves the performances. From our experiments, we observe that this is always the case for $\sigma \leq 32$. In order to speed up order basis implementations, we therefore use a x^{32}-adic version of our algorithms. For PM-Basis this corresponds to stopping the recursion at $\sigma = 32$, while for oPM-Basis this comes down to incrementing k by 32 in the "for" loop. For both of them, the calls of Basis are then replaced with M-Basis on input series mod x^{32}.

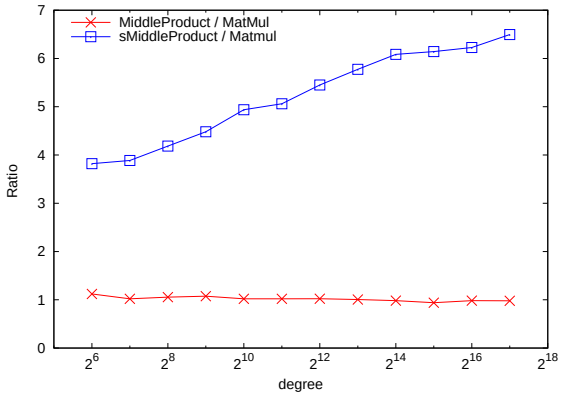

Figure 2: Relative performance of different middle products against polynomial matrix multiplication over $\mathbb{F}_{7340033}[x]^{16\times 16}$.

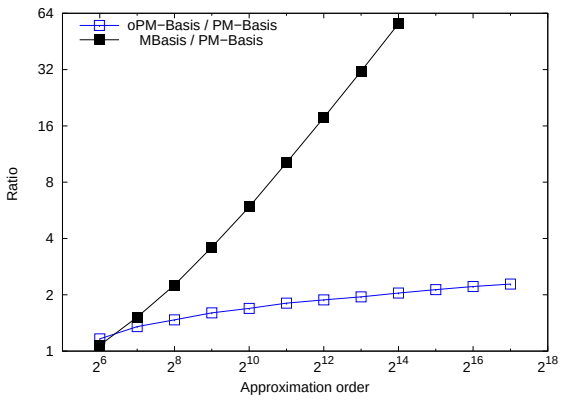

Figure 3: Relative performance of different online order basis algorithms against polynomial matrix multiplication. Matrix power series lies in $\mathbb{F}_{7340033}[x]^{32\times 16}$.

In figure 3, we report the relative performance of online order basis, *i.e.* M-Basis and oPM-Basis, against the fast offline order basis PM-Basis. From this figure, one can first see that quadratic approach, as M-Basis , is not competitive w.r.t. its fast counterpart as soon as $\sigma > 128$. The high efficiency of our polynomial matrix operations (multiplication and middle product) makes it possible to use fast variant

very early. Furthermore, the larger the matrix dimensions the earlier fast variants outperform M-Basis.

As shown in Figure 3, oPM-Basis almost keeps up with the performances of PM-Basis, The ratio of timings stays below 2 in the given range of orders, despite its theoretical logarithmic behavior. This makes our oPM-Basis algorithm an interesting approach when online computation makes sense, for example when computing the matrix power series coefficients is very expensive.

7.4 Block Wiedemann algorithms

As demonstrated in section 6, using early strategy in block Wiedemann should benefit in practice from online order basis calculation. In order to exhibit such a behavior in practice, we compare the timings of computing both the required sequence element UA^iV and the corresponding order basis for any approximation order up to a given bound. We characterize which algorithm is best suited depending on how early the algorithm terminates.

We use a random sparse matrix $A \in \mathrm{GL}_{2^{17}}(\mathbb{F}_{6946817})$ with 20 elements per row and two random dense matrices $U, V^T \in \mathcal{M}_{16\times 2^{17}}(\mathbb{F}_{6946817})$. To give a fair benchmark, we perform the calculation of the sequence elements UA^iV in parallel (on the column of V) while order basis calculation is done sequentially. Order basis could have also benefit from parallel computation but this only affects the result by a constant factor and does not change our observation.

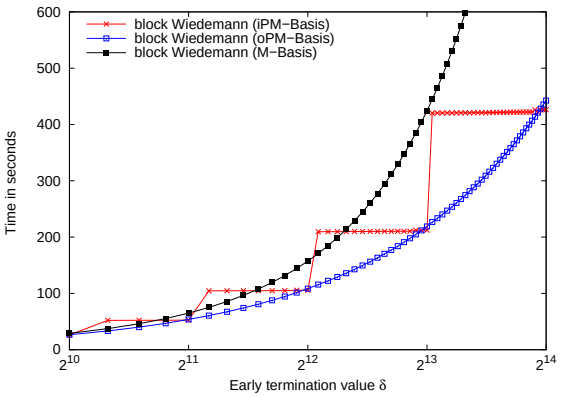

Figure 4: Computation times of early termination in block Wiedemann using order basis algorithms. $A \in \mathrm{GL}_{2^{17}}(\mathbb{F}_{6946817})$ **with** 20 **elts/row,** $m = n = 16$**.**

Figure 4 illustrates the measured times for the first two steps of block Wiedemann with early termination using the offline iPM-Basis and online M-Basis, oPM-Basis algorithms. Note that $\sigma = 2^{14}$ corresponds to the classic bound $(2N/m)$.

One sees on this figure that whenever early termination leads us to stop at a value δ smaller than the *a priori* bound σ_C, online algorithms can improve performances. Of course, this is only the case when δ is not close to a power of two, where iPM-Basis still remains the fastest. In other cases, M-Basis improves performance only for very small approximant orders, which is an implication of Proposition 10. As soon as δ is getting larger, the quadratic complexity of M-Basis makes it useless to defeat iPM-Basis even with its extra UA^iVs.

In the case of oPM-Basis, most of the possible values for δ allow to be faster than using the fast offline approach. Indeed, it is almost always true that the extra $\log(\delta)$ of oPM-Basis is negligible compared to few extra UA^iV needed by iPM-Basis. Except for very few cases around power of two, this makes oPM-Basis always the fastest algorithm, as predicted by Proposition 10.

8. REFERENCES

[1] B. Beckermann and G. Labahn. A uniform approach for the fast computation of matrix-type Padé approximants. *SIAM Journal on Matrix Analysis and Applications*, 15(3):804–823, 1994.

[2] B. Beckermann, G. Labahn, and G. Villard. Shifted normal forms of polynomial matrices. In *ISSAC'99*, p. 189–196. ACM, 1999.

[3] J. Berthomieu, J. v. d. Hoeven, and G. Lecerf. Relaxed algorithms for p-adic numbers. *J. Théor. Nombres Bordeaux*, 23(3):541–577, 2011.

[4] D. Coppersmith. Solving Homogeneous Linear Equation Over GF(2) via Block Wiedemann Algorithm. *Mathematics of Computation*, 62(205):333–350, 1994.

[5] J.-G. Dumas and P. Giorgi and C. Pernet. Dense Linear Algebra over Word-Size Prime Fields: The FFLAS and FFPACK Packages. *ACM Trans. Math. Softw.*, 35(3):19:1–19:42, 2008.

[6] P. Giorgi, C.-P. Jeannerod, and G. Villard. On the complexity of polynomial matrix computations. In *ISSAC'03*, p. 135–142. ACM, 2003.

[7] F. C. Hennie. On-line turing machine computations. *Electronic Computers, IEEE Transactions on*, EC-15(1):35 –44, 1966.

[8] J. v. d. Hoeven. Relax, but don't be too lazy. *J. Symbolic. Comput.*, 34(6):479–542, 2002.

[9] J. v. d. Hoeven. Relaxed multiplication using the middle product. In *ISSAC '03*, p. 143–147. ACM, 2003.

[10] C.-P. Jeannerod, C. Pernet, and A. Storjohann. Rank-profile revealing gaussian elimination and the cup matrix decomposition. *J. Symbolic Comput.*, 56:46–68, 2013.

[11] C.-P. Jeannerod and G. Villard. Asymptotically fast polynomial matrix algorithms for multivariable systems. *J. Symbolic. Comput.*, 79(22):1359–1367, 2006.

[12] E. Kaltofen and W.-S. Lee. Early termination in sparse interpolation algorithms. *J. Symbolic. Comput.*, 36(3-4):365–400, 2003.

[13] E. Kaltofen and G. Villard. On the complexity of computing determinants. *Comput. Complexity*, 13(3-4):91–130, 2004.

[14] E. Kaltofen and G. Yuhasz. On the matrix berlekamp-massey algorithm. *ACM Trans. on Algorithms*, 2013. To appear.

[15] E. Thomé. Subquadratic computation of vector generating polynomials and improvement of the block Wiedemann algorithm. *J. Symbolic Comput.*, 33(5):757–775, July 2002.

[16] W. J. Turner. *Black Box Linear Algebra with the LinBox Library*. PhD thesis, North Carolina State University, 2002.

[17] W. J. Turner. A block wiedemann rank algorithm. In *ISSAC '06*, p. 332–339. ACM, 2006.

[18] G. Villard. A study of coppersmith's block wiedemann algorithm using matrix polynomials. Technical report, Research Report 975 IM, LMC-IMAG, 1997.

[19] W. Zhou. *Fast Order Basis and Kernel Basis Computation and Related Problems*. PhD thesis, Univ. of Waterloo, 2013.

[20] W. Zhou and G. Labahn. Efficient algorithms for order basis computation. *J. Symbolic Comput.*, 47(7):793 – 819, 2012.

[21] W. Zhou, G. Labahn, and A. Storjohann. Computing minimal nullspace bases. In *ISSAC '12*, p. 366–373. ACM, 2012.