Online order basis algorithm and its impact on the block Wiedemann algorithm

Pascal Giorgi, Romain Lebreton

Online order basis algorithm and its impact on the block Wiedemann algorithm*

Pascal Giorgi†, Romain Lebreton‡

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 performance 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 the determinant, column reduction [8] or minimal nullspace basis [24] has reduced these problems to the simpler computation of polynomial matrix multiplication. A complete panel of these reductions can be found in [13, 22].

Since their introduction in 1994 [1], order basis computations have a quasi-optimal algebraic cost in the order of approximation. Fast linear algebra has been included later on for square matrices [8], then for any matrix dimensions [23]. Historically, order basis were introduced to generalize Hermite-Padé approximants to 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, 21] 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 [16] for a classification of all possible algorithms with their corresponding reference and complexity.

*This work has been supported by the French ANR under the grants HPAC (ANR-11-BS02-013), CATREL (ANR-12-BS02-001).

†Université Montpellier 2 - CNRS, LIRMM

‡Université Montpellier 2 - CNRS, LIRMM; and University of Waterloo

Modern sparse linear algebra over a finite field relies on Coppersmith's block Wiedemann method [5], which in turn computes a minimal matrix generating polynomial. This is the case for linear systems solving [5] or for rank computations [19]. For generic matrices with low rank deficiency, as in integer factorization, the general *a priori* bound on the degree of this polynomial is tight. However, a significant rank deficiency makes this bound loose, which introduces a computational overhead. This situation has been observed in [6] when computing the rank of matrices from Algebraic K-theory using the block Wiedemann method of [19]. In such a case, early termination techniques [14] are an important tool to probabilistically detect the degree of the minimal matrix generating polynomial. However no fast block Wiedemann method enables early termination in the order basis computation to the best of our knowledge.

Hence we focus in this paper on the application of order basis to the block Wiedemann framework in the context of early termination. In particular, existing fast order basis algorithms such as Algorithm PM-Basis of [8] does not allow for early termination.

Main results and outline of the paper. We contribute by giving two new algorithms, called iPM-Basis and oPM-Basis, which both have a quasi-linear time complexity in the degree of the approximation's order. After some background on order basis in Sections 2 and 3, we introduce in Section 4 Algorithm iPM-Basis, which is an iterative variant of PM-Basis more suited to early termination. This enables us to incorporate in block Wiedemann both early termination and quasi-linear time algorithm for the computation of the minimal matrix generating polynomial.

However, this is not optimal yet because we may have to compute more coefficients of the input of the order basis than necessary. Besides, the cost of computing this input is dominant in block Wiedemann, and so it is of great interest to minimize the number of required coefficients. Since *online* algorithms require minimal knowledge on the input, we propose a fast online order basis oPM-Basis in Section 5.

Then we study our order basis algorithms inside block Wiedemann method in the context of early termination. We analyze the complexity of our approach in Section 6 and make explicit the potential gains in Proposition 7. Finally Section 7 gives implementations and benchmarks and shows the positive impact of oPM-Basis in practice (see Figure 4).

2 Background

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 $B \in \mathbb{K}[x]^{m\times n}$ such that $\deg(A) < d$, $\deg(B) < h$ and $d \leqslant h$. 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 Row reduced matrix

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 $\vec{s} = [s_1, \ldots, s_n] \in \mathbb{Z}^n$ corresponds to $\max_{1\leqslant i\leqslant n}(s_i + \deg v[i]) \in \mathbb{Z}$. This naturally extends to matrices $A \in \mathbb{K}[x]^{m\times n}$ by considering the vector in $\mathbb{Z}^m$ of $\vec{s}$-row degrees of the rows of A.

A matrix $R \in \mathbb{K}[x]^{m\times n}$ is said to be $\vec{s}$-row reduced if the $\vec{s}$-row degree of R is less than or equal to the $\vec{s}$-row degree of UR for any unimodular matrix U. Comparisons of $\vec{s}$-row degrees use the lexicographic order on their sorted vectors. The existence of a minimal $\vec{s}$-row degree matrix in the set of UR for U unimodular is ensured by [22, Lemma 2.11]. One can ensure the unicity of row reduction by using the well known Popov form [2].

The following lemma gives a criterion to certify that a matrix is $\vec{s}$-row reduced. Let us denote by $x^{\vec{s}}$ the diagonal matrix $Diag(x^{s_1}, \ldots, x^{s_n}) \in \mathbb{K}[x]^{n\times n}$.

Lemma 1 ([2, 22]). *Let $\vec{u}$ be the $\vec{s}$-row degree of a matrix $R \in \mathbb{K}[x]^{m\times n}$ and $T \in \mathbb{K}^{m\times n}$ such that $x^{-\vec{u}}Rx^{\vec{s}} = T + O(x^{-1})_{x\to\infty}$. Then R is $\vec{s}$-row reduced if and only if T has full rank.*

The following lemma states the compatibility of $\vec{s}$-row degree and row reducedness with matrix multiplication.

Lemma 2 ([22, Lemmas 2.14, 2.18]). *Let A, B belong to $\mathbb{K}[x]^{m\times m}$ with B invertible, $\vec{u}$-row reduced and of $\vec{u}$-row degree $\vec{v}$. Then the $\vec{u}$-row degree of AB is equal to the $\vec{v}$-row degree of A and $AB \in \mathbb{K}[x]^{m\times m}$ is $\vec{u}$-row reduced if and only if $A \in \mathbb{K}[x]^{m\times m}$ 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 $m \geqslant n$, σ a positive integer and (F, σ) be the $\mathbb{K}[x]$-module $\left\{v \in \mathbb{K}[x]^{1\times m} \text{ s.t. } vF \equiv 0 \bmod x^\sigma\right\}$. A polynomial matrix P is a *(left) order basis* of F of order $\sigma \in \mathbb{N}^*$ and shift $\vec{s} \in \mathbb{Z}^m$ if the rows of P form a basis of (F, σ) and P is $\vec{s}$-row reduced (see [22, 23] for more details). Without loss of generality, we only consider in this paper the case $n = \Theta(m)$ as in [8]. Indeed the techniques of [23] 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 [8] using the framework of order basis of [22].

Throughout the paper, the order basis algorithms, except Basis, will take as input a power series matrix $F \in \mathbb{K}[[x]]^{m\times n}$, a shift vector $\vec{s} \in \mathbb{Z}^m$ 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 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 a row 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 [12] 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 row echelon form. Note that among all possible PLE decompositions, we use one revealing the row rank profile.

Let ω be the exponent in the complexity of matrix multiplication over fields, which satisfies $\omega < 2.3727$ [20].

Algorithm 1: Basis$(F_0, \vec{s})$

Input: $F_0 \in \mathbb{K}^{m \times n}$, $\vec{s} \in \mathbb{Z}^m$
Output: $M \in \mathbb{K}[x]^{m \times m}$, $\vec{u} \in \mathbb{Z}^m$

1: Find a permutation π s.t. $\pi\vec{s}$ is sorted increasingly
2: Compute the PLE decomposition τLE of πF_0
3: Let $r = \texttt{rank}(E)$ 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 1. *Algorithm Basis outputs a $(1, \vec{s})$ order basis M of F_0 and its $\vec{s}$-row degree $\vec{u}$ in $O(mnr^{\omega-2})$ operations in $\mathbb{K}$, where r corresponds to the rank of F_0.*

Proof. For the correctness of the algorithm, we need to prove that $MF_0 \equiv 0 \bmod x$, $\det M \neq 0$ 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_0 are also zero. Since the first r rows of M are $[xI_r\ 0]$, it is clear that $MF_0 \equiv 0 \bmod x$. 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 $\vec{u} = [s_1+1, \ldots, s_r+1, s_{r+1}, \ldots, s_m]$ is the $\vec{s}$-row degree of M. Therefore, we have $x^{-\vec{u}} M x^{\vec{s}} = I_m + O(x^{-1})_{x \to \infty}$, which implies by Lemma 1 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 [12]. □

3.2 Quadratic algorithm

To compute a $(\sigma, \vec{s})$-order basis, Algorithm M-Basis presented in [8] works iteratively by induction on the approximation order.

Proposition 2. *Algorithm M-Basis is correct and runs in $O(m^\omega \sigma^2)$ 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.

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

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

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** $P_{\sigma-1}, \vec{u}_{\sigma-1}$

Assume P_{k-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 needs 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 $\vec{u}_{k-1}$. Similarly, M_k is $\vec{u}_{k-1}$-row reduced and its $\vec{u}_{k-1}$-row degree is $\vec{u}_k$. So it follows directly from Lemma 2 that P_k is $\vec{s}$-row reduced and its $\vec{s}$-row degree is $\vec{u}_k$.

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 into the 'for' loop gives the announced complexity for M-Basis. □

3.3 Quasi-linear algorithm

A divide-and-conquer approach has been presented in [8], namely the PM-Basis algorithm. The idea is similar to the ones used in [1, 17]: it reduces the computation of one order basis of order σ to two order bases of order about $\sigma/2$.

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

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

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 [8], 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 Lemma 2. 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 $k = \sum_{i=1}^{r} 2^{n_i}$ be the binary decomposition of k with $n_1 < \cdots < n_r$. The valuation $\nu_2(k)$ in 2 of positive k corresponds to n_1. By convention, $\nu_2(0) = +\infty$.

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

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

1: $F^{(\infty)} = F$
2: $M_0, \vec{u}_0 = \mathsf{Basis}(F \bmod x, \vec{s})$
3: **for** $k = 1$ **to** $\sigma - 1$ **do**
4: $\quad v = \nu_2(k)$, $v' = \nu_2(k - 2^v)$
5: $\quad M^{(v)} = ((M_{k-1} \cdot M^{(0)}) \cdot M^{(1)}) \cdots M^{(v-1)}$
6: $\quad F^{(v)} = \mathrm{MP}(M^{(v)}, F^{(v')}, 2^v + 1, 2^{v+1})$
7: $\quad M_k, \vec{u}_k = \mathsf{Basis}(F^{(v)} \bmod x,\ \vec{u}_{k-1})$
8: Let $\sum_{i=1}^{r} 2^{n_i}$ be the binary decomposition of σ
9: $M^{(n_1)} = ((M_{\sigma-1} \cdot M^{(0)}) \cdot M^{(1)}) \cdots M^{(n_1-1)}$
10: **return** $(M^{(n_1)} \cdot M^{(n_2)}) \cdots M^{(n_r)}, \vec{u}_{\sigma-1}$

It is easy to add a stopping 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. Besides, our algorithm should store all the M_k in one variable M and all the $\vec{u}_k$ in $\vec{u}$ but these indexed variables will be useful in the proof.

Before proving that iPM-Basis is the iterative counterpart of PM-Basis, we give some hindsight on iPM-Basis. The polynomial matrices M_k (of degree at most 1) coincide with those of M-Basis. Our order basis output consists in their product $M_{\sigma-1} \cdots M_0$, which is computed using the divide-and-conquer scheme of PM-Basis. For this matter, we have the variables $M^{(v)}$ which store at step k the product $M_{r-1} \cdots M_{r-2^v}$ where $r = k - (k \bmod 2^v)$. We perform the same products than a binary multiplication tree on $M_0, \ldots, M_{\sigma-1}$ would do when σ is a power of two. But our approach decomposes these computations in steps and therefore yields an iterative variant of the binary multiplication tree recursive algorithm.

Proposition 3. *For any $F, \vec{s}$ and $t \in \mathbb{N}$, Algorithms PM-Basis and iPM-Basis have the same output and perform the same computations on the input $F, 2^t, \vec{s}$.*

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

Now recursively, suppose the result true for inputs $F, 2^r, \vec{s}$ with $r < t$ and let us prove it for the input $F, 2^t, \vec{s}$. The program $\mathsf{PM}_t :=$ PM-Basis$(F, 2^t, \vec{s})$ is made of 4 instructions, and we claim that they are mapped to subparts of $\mathsf{It}_t :=$ iPM-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. lines 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 $M^{(\sigma)}$.

Proof of 1. First, the call to PM_{t-1} on line 4 of PM_t corresponds to It_{t-1} by the inductive 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 $M^{(t-1)}$ of line 9, which corresponds to the product of It_t, line 5 for $k = 2^{t-1}$.

Proof of 2. Let us prove that the inputs of both middle products are the same. First, $M^{(t-1)}$ and P_l are equal because they are the respective outputs of It_{t-1} and PM_{t-1} which coincide by the inductive hypothesis. Concerning the second input, $F^{(v')} = F^{(\infty)}$ is equal to F because $F^{(\infty)}$ has not been overwritten since its definition $F^{(\infty)} = F$ at line 1. So the middle products coincide and their outputs F' and $F^{(t-1)}$ correspond.

Proof of 3. By recurrence hypothesis, line 6 of PM_t corresponds to $\mathsf{It}'_{t-1} := \mathsf{iPM\text{-}Basis}(F^{(t-1)}, 2^{t-1}, \vec{u}_{2^{t-1}-1})$. Thus, we will prove the correspondence of It'_{t-1} followed by line 7 of PM_t, with $\mathsf{It}_{t,\mathrm{end}}$. We split both of these programs in 4 parts: 1) we will see that the initialization $F^{(\infty)} = F^{(t-1)}$ of line 1 of It'_{t-1} is compensated by direct access to $F^{(t-1)}$ instead of $F^{(\infty)}$ in $\mathsf{It}_{t,\mathrm{end}}$; 2) the initialization of 2 of It'_{t-1} corresponds to line 7, $k = 2^{t-1}$ of It'_{t-1}; 3) we will show later that respective 'for' loops of the two programs correspond; 4) 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.

Let us prove that both respective 'for' loops of It'_{t-1} and $\mathsf{It}_{t,\mathrm{end}}$ read the same variables. Let $1 \leqslant k \leqslant 2^{t-1} - 1$ be the indices of the first 'for' loop and $k' = k + 2^{t-1}$ the corresponding indices inside $\mathsf{It}_{t,\mathrm{end}}$. Notice that $v = \nu_2(k) = \nu_2(k')$. Similarly $v' = \nu_2(k - 2^v) = \nu_2(k' - 2^v)$ except if $k = 2^r$ in which case $\nu_2(k - 2^v) = \infty$ and $\nu_2(k' - 2^v) = t - 1$. Therefore the variables $(M^{(i)})_{0 \leqslant i \leqslant v}$ and $F^{(v')}$ that both program read coincide. Note that the variables $F^{(v')}$ also coincide when $k = 2^r$ because they both point to $F^{(t-1)}$. □

Corollary 1. *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. Proposition 3 proves both the complexity and correctness statements of iPM-Basis, but only for orders σ that are powers of two. We deduce our complexity estimate for general σ from the case $\sigma = 2^r$. Concerning correctness, we will prove that we match the output of M-Basis. The variables M_k and $\vec{u}_k$ of iPM-Basis coincide with those of M-Basis because PM-Basis and iPM-Basis perform the same computations when $\sigma = 2^r$ and PM-Basis performs the same calls to Basis as M-Basis.

It remains to prove that we output $M_{\sigma-1} \cdots M_0$ like M-Basis to conclude. Let $\sigma = \sum_{i=1}^{r} 2^{n_i}$ be the binary decomposition of σ with $\nu_2(\sigma) = n_1 < \cdots < n_r$. 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. Then at the end of the algorithm, the variables $M^{(n_i)}$ equal to $M_{\varphi(k,r-i+1)-1} \cdots M_{\varphi(k,r-i)}$ when $1 \leqslant i \leqslant r$ and their product is $M_{\sigma-1} \cdots M_0$ as claimed. □

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. Moreover, during the intermediate steps of M-Basis$(F, \sigma, \vec{s})$, only $F \bmod x^{k+1}$ is read to compute M_k. Therefore M-Basis allows for savings both on the input and on the computations in case of early termination.

This is not the case for the fast algorithms of the family of PM-Basis. The speed of these algorithms is attained by performing computations in advance, which in turn require more knowledge on F. Consider the computation PM-Basis$(F, 2^r, \vec{s})$ and let us focus on its intermediate steps which are made explicit in iPM-Basis. Then for any $t \leqslant r$ and any intermediate step k such that $2^{t-1} < k \leqslant 2^t$, Algorithm iPM-Basis reads $F \bmod x^{2^t}$ to compute M_k (see line 6 for $k = 2^{t-1}$ where $F^{(v')} = F$).

In some applications, such as the block Wiedemann method, the cost of computing the input F is dominant (see Section 6) and minimizing its required precision can have a practical impact. The purpose of this section is to give an algorithm that requires minimal knowledge on F at each intermediate step 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, i be one of its inputs and o its only output. Assume that both $i = [i_0, \ldots, i_n]$ and $o = [o_0, \ldots, o_n]$ are given as 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 [9], was popularized in the domain of Computer Algebra in [10] 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 $[F_i]_{i<\sigma}$ and the output to be the sequence $[M_i]_{i<\sigma}$. 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.

We have seen that M-Basis is an online order basis algorithm. Our objective in this section is to derive a fast online order basis algorithm from iPM-Basis. We have noticed that the middle product of step 6 of iPM-Basis may read more entries of F than necessary. Therefore, we need to perform this step differently.

5.2 Shifted online middle product

Throughout this section, we will denote by R a ring and M a left R-module. Let $A \in R[x]$ and $B \in M[x]$ with A of degree less than d. This theoretical framework is meant to allow for the multiplication of A by B in our special case $R = \mathbb{K}^{m \times m}$ and $M = \mathbb{K}^{m \times n}$.

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 case $h = 2d-1$ for the rest of this section, where $\deg(\mathrm{MP}(A, B, d)) < d$. As in the *offline* (not online) case, we have a naive algorithm for the middle product which consists in computing $(A \cdot B) \bmod x^{2d-1}$ and removing the unnecessary lower terms. We denote by $\mathsf{H}(d)$ the number of R-module operations on M necessary to multiply two power series at precision d (*i.e.* modulo x^d) by a half-line algorithm. Similarly to offline multiplication, we suppose that H is super-linear, *i.e.* that $r\mathsf{H}(s) \leqslant \mathsf{H}(rs)$ for any $r, s \in \mathbb{N}$. The cost of the naive shifted online middle product algorithm is therefore $\mathsf{SMP}_{\mathsf{Naive}}(d) := \mathsf{H}(2d-1)$.

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 5: sMiddleProduct(A, B, d)

Input: $A \in R[x]$ of degree $< d$, $B \in M[x]$
Output: $\mathrm{MP}(A, B, d) \in M[x]$

1: $C = (A_{0\ldots d} \cdot B_{0\ldots d}) \operatorname{div} x^{d-1}$
2: **for** $k = 1$ **to** $d-1$ **do**
3: $\quad m = \nu_2(k)$
4: $\quad C = C + \mathrm{MP}(A_{0\ldots 2^{m+1}-1}, B_{d+k-2^m\ldots d+k}, 2^m)\ x^k$
5: **return** C

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

Let $B' := B_{d\ldots 2d-1}$. For the computation of U, the reading constraint translates into 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 $\mathsf{H}(d-1)$. Finally the addition $C = L + xU$ costs $O(d)$. Altogether, Algorithm requires $\mathsf{SMP}_{\mathsf{GL}}(d)$ R-module operations on M where $\mathsf{SMP}_{\mathsf{GL}}(d) := \mathsf{M}(d) + \mathsf{H}(d-1) + O(d)$. Since $d \ll \mathsf{M}(d) \ll \mathsf{H}(d)$ and $2\mathsf{H}(d-1) \leqslant \mathsf{H}(2d-1)$, we deduce

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

In Figure 1 and Algorithm sMiddleProduct (see lines 3-4), we use the half-line multiplication of [11] due to its good performances.

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

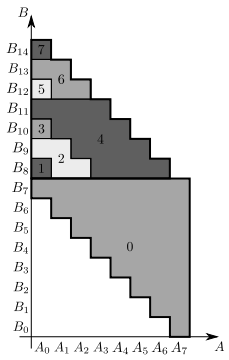

Figure 1: Shifted online middle product $\mathrm{MP}(A, B, 8)$

apply; we can cut the middle product into two parts, and then 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 [3, 10]. In this data structure, polynomials are given as a promise, and the evaluation of its coefficients is delayed.

We assume that we dispose of a class $O_{\mathsf{sMiddleProduct}}$ that performs the shifted online middle product using the lazy data structure. In this setting, step k of Algorithm sMiddleProduct is performed when the kth coefficient of the middle product is asked for. By computing the middle product in this fashion, we obtain an online order basis algorithm oPM-Basis.

Algorithm 6: oPM-Basis$(F, \sigma, \vec{s})$

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

1: $F^{(\infty)} = F$
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: $\quad M^{(v)} = ((M \cdot M^{(0)}) \cdot M^{(1)}) \cdots M^{(v-1)}$
6: $\quad F^{(v)} = O_{\mathsf{sMiddleProduct}}(M^{(v)}, F^{(v')}, 2^v + 1, 2^{v+1})$
7: $\quad M, \vec{u}_k = \mathsf{Basis}(F^{(v)} \bmod x,\ \vec{u}_{k-1})$
8: Let $\sum_{i=1}^{r} 2^{n_i}$ be the binary decomposition of σ
9: $M^{(n_1)} = ((M \cdot M^{(0)}) \cdot M^{(1)}) \cdots M^{(n_1-1)}$
10: **return** $(M^{(n_1)} \cdot M^{(n_2)}) \cdots M^{(n_r)}, \vec{u}_{\sigma-1}$

Proposition 4. *Algorithm oPM-Basis is correct and online. It runs in* $O(m^\omega \mathsf{M}(\sigma) \log^2(\sigma))$ *operations in* $\mathbb{K}$.

Proof. Noticing that $F^{(v)} = (M_{k-1} \cdots M_0 F)_{k \ldots k+2^v}$, we deduce that the call to $(F^{(v)} \bmod x)$ on line 7 will read at most the entries $F_0, \ldots, F_k$ of F at step k.

The cost analysis is similar to the one of PM-Basis, except that middle products now cost $\mathsf{H}(d) = O(\mathsf{M}(d)\log(d))$ instead of $\mathsf{M}(d)$ [10]. Thus, Algorithm oPM-Basis requires $O(m^\omega \mathsf{H}(\sigma)\log(\sigma)) = O(m^\omega \mathsf{M}(\sigma)\log^2(\sigma))$ operations in $\mathbb{K}$, which is quasi-linear in σ. □

6 Application to block Wiedemann

Let $A \in \mathbb{K}^{N \times N}$ be a sparse matrix, which means that it has $O(N \log(N)^C)$ non-zero entries for a constant $C > 0$. Block Wiedemann methods are useful for handling such matrices, and especially for linear systems solving [5] or for computing their rank [19]. The main ingredient of block Wiedemann algorithms is the computation of the left minimal matrix generating polynomial $\Pi_U^{AV} \in \mathbb{K}[x]^{m \times m}$ of the matrix sequence $S = (S_i)_{i \in \mathbb{N}}$ where $S_i = (UA^iV) \in \mathbb{K}^{m \times n}$ and $m \geqslant n$. Here $U \in \mathbb{K}^{m \times N}$ and $V \in \mathbb{K}^{N \times n}$ are chosen randomly. Note that A can be preconditioned to ensure structural properties (e.g. generic rank profile, no nilpotent blocks of size greater than one in its Jordan form) [4] and then allow a good probability of success for block Wiedemann method.

Definition 1. *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*

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

and $\det(\Pi) \neq 0$. *It is said to be* minimal *if it is row reduced and* $\deg \det(\Pi)$ *is minimal. Furthermore, it is* unique *if it is in Popov form [16].*

As described in [18, 21], 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's work [5] 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 rank deficient matrices, as experimented in [6].

Theorem 2.12 of [15] 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 a minimal matrix generating polynomial of S can be derived from an order basis at precision $\sigma_{\mathsf{KV}} = \lceil \mu/m \rceil + \lceil \mu/n \rceil + O(1)$ [15, 19]. Note that μ gives a bound on the degree of the determinant of Π_U^{AV}.

When the values of μ and σ_{KV} are not known, and assuming $\mu \ll N$, early termination is a good strategy to stop the course of the order basis algorithm. We use a 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 condition is similar to getting consecutive zero discrepancies in the Berlekamp-Massey algorithm [14].

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

following, we only consider the sequential case as the analysis is equivalent in the parallel setting. Indeed, using t processors with t divides m will divide the cost for computing the sequence elements by t. The gain is similar for parallel computation of order basis.

We demonstrate below that the 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 is always the case with oPM-Basis whenever N is big. For the sake of simplicity we now assume that $m = n$.

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 $2\mu/m < \delta \leqslant \sigma_{\mathsf{KV}} \leqslant \sigma_{\mathsf{C}}$ [15]. We will assume that our heuristic stops 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 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). We denote these complexities respectively by $\mathcal{C}_{\text{M-Basis}}$, $\mathcal{C}_{\text{iPM-Basis}}$ 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 $2^{t-1} < \delta \leqslant 2^t$. Let $\mathcal{X}$ be the cost of computing one coefficient of the sequence F, *i.e.* $UA^iV = U(A(A^{i-1}V))$, that is $\mathcal{X} = \Theta(mN(m + \log(N)^C))$. Then

$$\begin{aligned}
\mathcal{C}_{\text{M-Basis}} &= \delta\mathcal{X} + O(m^\omega \delta^2), \\
\mathcal{C}_{\text{iPM-Basis}} &= 2^t\mathcal{X} + O(m^\omega \delta \log(\delta)), \\
\mathcal{C}_{\text{oPM-Basis}} &= \delta\mathcal{X} + O(m^\omega \delta \log^2(\delta)).
\end{aligned}$$

Proposition 5. *The ratio $\mathcal{C}_{\text{iPM-Basis}}/\mathcal{C}_{\text{M-Basis}}$ tends to $2^t/\delta$ when μ/N tends to zero.*

Proof. First, one has $\frac{\mathcal{C}_{\text{iPM-Basis}}}{\mathcal{C}_{\text{M-Basis}}} = \frac{2^t/\delta + O(\varepsilon_1)}{1 + O(\varepsilon_2)}$ 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 $\mathcal{C}_{\text{iPM-Basis}}/\mathcal{C}_{\text{M-Basis}}$ tends to $2^t/\delta$ when μ/N tends to 0. Using $\mathcal{X} = \Omega(m^2 N)$ and $\delta = O(\mu/m)$, we get that $\varepsilon_2 = O(m^{\omega-3}\mu/N)$ which tends to 0 as claimed. □

Proposition 6. *The ratio $\mathcal{C}_{\text{iPM-Basis}}/\mathcal{C}_{\text{oPM-Basis}}$ 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 5 and 6 state that we gain a constant factor $K_\delta = 2^t/\delta$ compared to offline block Wiedemann. This constant depends on δ and satisfies $1 \leqslant K_\delta < 2$. The upper bound is tight as it is easy to find δ such that K_δ is arbitrarily close to 2. The following proposition states that the average gain is larger than 1.

Proposition 7. *Let δ be a random variable uniformly distributed between 1 and $\sigma_C = 2^r$. 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, the mean value of K_δ would be

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

The actual mean value of K_δ is

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

$\square$

7 Practical performance

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 motivation for 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 fields [7]. Second, we provide cache friendly DFT transforms for polynomials, which use 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, namely matrix of polynomials as well as polynomial of matrices, and efficient conversions between them.

Table 1 gives a quick preview of the performance of our code. For comparison purposes, we provide the times of two standard libraries: FLINT (www.flintlib.org) and MATHEMAGIX (www.mathemagix.org). Our code provides similar or better performance than these reference libraries. In particular, our optimizations on DFT are close to the ones of MATHEMAGIX, which are 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 MATHEMAGIX does not yet provide such a support in their `matrix_naive` type. This emphasizes the benefit of the `FFLAS` approach, which always makes use of the most recent technology. Regarding FLINT's performance, we explain the gap in performance by our use of both cache friendly storage and vectorization.

$m, \log_2(d)$	MATHEMAGIX	FLINT 2.4	our code
$16, 10$	**0.02s**	$0.26s$	$0.03s$
$16, 11$	**0.04s**	$0.70s$	$0.06s$
$16, 12$	**0.09s**	$1.68s$	$0.13s$
$16, 13$	**0.20s**	$4.52s$	$0.28s$
$128, 9$	$1.00s$	$26.21s$	**0.82s**
$256, 8$	$4.00s$	$36.71s$	**1.75s**
$512, 9$	$69.19s$	$465.66s$	**19.64s**
$1024, 6$	$71.36s$	$115.52s$	**13.95s**
$2048, 5$	$298.27s$	$263.88s$	**48.90s**

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

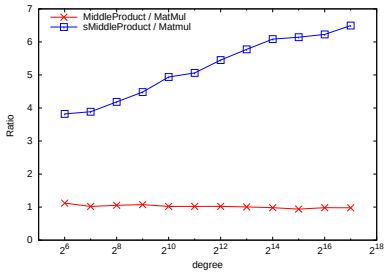

Figure 2: Relative performance of two middle products against polynomial matrix multiplication over $\mathbb{F}_p[x]^{16\times 16}$ for p a 23-bit prime.

7.2 Online middle product

Our fast shifted online middle product relies on the half-line product of [11]. Such a 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 the offline and shifted online 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 $1/4 \cdot \log(\sigma)$.

7.3 Online order basis

As described in Sections 3 to 5, fast order basis algorithms reduce to polynomial matrix multiplication. Our implementations of iPM-Basis and oPM-Basis follow this reduction and therefore mainly use the online middle products of Section 7.2 and the polynomial matrix multiplication of Section 7.1. In practice, for small values of σ, it appears not surprisingly that using a fast order basis algorithm is not relevant, and that switching to a quadratic one as M-Basis improves the performance. From our experiments, we observe that this is always the case for $\sigma \leqslant 32$. In order to speed up order basis implementations, we therefore use an x^{32}-adic version of our algorithms. For iPM-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 to Basis are then replaced

with calls to M-Basis on the input series mod x^{32}.

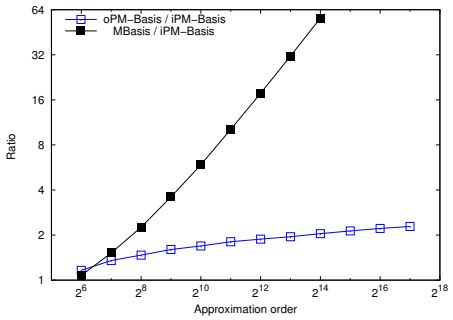

Figure 3: Relative performance of two online order basis algorithms against polynomial matrix multiplication. The input matrix power series are in $\mathbb{F}_p[x]^{32\times 16}$ for p a 23-bit prime.

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 iPM-Basis. From this figure, one can first see that a quadratic approach, such 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 the 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 performance of iPM-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 termination 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 elements 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 $A \in \mathcal{M}_{2^{17}}(\mathbb{F}_{6946817})$ 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 columns of V) while order basis calculation is done sequentially. Order basis could also benefited from parallel computations but this only affects the result by a constant factor and does not change our observation.

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 performance. Of course, this is only the case when δ is not close to a power of

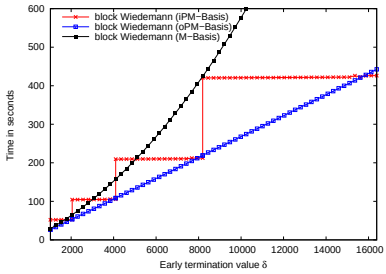

Figure 4: Timings of block Wiedemann with early termination for different order basis algorithms. $A \in \mathcal{M}_{2^{17}}(\mathbb{F}_p)$ with 20 elts/row, $m = 16$ and p a 23-bit prime.

two, where iPM-Basis still remains the fastest. In other cases, M-Basis improves the performance only for very small approximant orders, which is a consequence of Proposition 5. As soon as δ is getting larger, the quadratic complexity of M-Basis is not competitive with the quasi-linearity of iPM-Basis, even with its extra UA^iV's.

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's needed by iPM-Basis. Except for very few cases around powers of two, this makes oPM-Basis always the fastest algorithm, as predicted by Proposition 6.

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] L. Chen, W. Eberly, E. Kaltofen, B. D. Saunders, W. J. Turner, and G. Villard. Efficient matrix preconditioners for black box linear algebra. *Linear Algebra and its Applications, 343-344:119–146, 2002.*

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

[6] J.-G. Dumas, P. Giorgi, P. Elbaz-Vincent, and A. Ubańska. Parallel computation of the rank of large sparse matrices from algebraic k-theory. In *PASCO'07*, p. 43–52. ACM, 2007.

[7] J.-G. Dumas, 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.

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

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

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

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

[12] 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.

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

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

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

[16] E. Kaltofen and G. Yuhasz. On the matrix Berlekamp-Massey algorithm. *ACM Trans. on Algorithms*, 9(4), 2013.

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

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

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

[20] V. Vassilevska Williams. Multiplying matrices faster than Coppersmith-Winograd. In *STOC'12*, p. 887–898. ACM, 2012.

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

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

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

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