Recurrence Width for Structured Dense Matrix Vector Multiplication

ALBERT GU[*] ROHAN PUTTAGUNTA[*] CHRISTOPHER RÉ[*] ATRI RUDRA[†]

[*] Department of Computer Science
Stanford University
{albertgu,rohanp,chrismre}@stanford.edu

[†] Department of Computer Science and Engineering
University at Buffalo, SUNY
atri@buffalo.edu

Abstract

Matrix-vector multiplication is one of the most fundamental computing primitives that has been studied extensively. Given a matrix $\mathbf{A} \in \mathbb{F}^{N\times N}$ and a vector $\mathbf{b} \in \mathbb{F}^{N}$, it is known that in the worst-case $\Theta(N^2)$ operations over $\mathbb{F}$ are needed to compute $\mathbf{Ab}$. Many classes of *structured dense* matrices have been investigated which can be represented with $O(N)$ parameters, and for which matrix-vector multiplication can be performed with a sub-quadratic number of operations. One such class of structured matrices that admit near-linear matrix-vector multiplication are the orthogonal polynomial transforms whose rows correspond to a family of orthogonal polynomials. Other well known classes include the Toeplitz, Hankel, Vandermonde, Cauchy matrices and their extensions (e.g. confluent Cauchy-like matrices) which are all special cases of a displacement rank property. In this paper, we identify a notion of recurrence width t of matrices $\mathbf{A}$ so that such matrices can be represented with t^2N elements from $\mathbb{F}$. For matrices with constant recurrence width we design algorithms to compute both $\mathbf{Ab}$ and $\mathbf{A}^T\mathbf{b}$ with a near-linear number of operations. This notion of width is finer than all the above classes of structured matrices and thus computes near-linear matrix-vector multiplication for all of them using the *same* core algorithm. Technically, our work unifies, generalizes, and (we think) simplifies existing state-of-the-art results in structured matrix-vector multiplication. We consider generalizations and variants of this width to other notions that can also be handled by the same core algorithms. Finally, we show how applications in disparate areas such as multipoint evaluations of multivariate polynomials and computing linear sequences can be reduced to problems involving low recurrence width matrices.

1 Introduction

1.1 Background and Overview of Our Results

In this paper, we focus on matrices in $\mathbb{F}^{N\times N}$, where $\mathbb{F}$ is any field, for which one can design a quasilinear time matrix-vector multiplication algorithm: i.e. an algorithm that takes $O(N\log^{O(1)} N)$ (which we will denote by $\tilde{O}(N)$) operations over $\mathbb{F}$. Such algorithms have been dubbed *superfast* algorithms in the matrix-vector multiplication literature [39]. Since superfast algorithms do not have time to read all $\Theta(N^2)$ elements of dense matrices, matrices that admit superfast multiplication algorithms must be structured in the sense of being expressible with a small number of parameters. Many problem such as the Discrete Fourier Transform, polynomial and rational multipoint evaluation/interpolation, and orthogonal polynomial projections can be expressed as matrix-vector multiplication involving dense, structured matrices, and specialized superfast algorithms have been designed for many of these specific problems [17, 22, 38]. We are interested in unifying these problems and their notions of structure. To this end, we introduce the concept of recurrence width, a new measure of the structural complexity of certain dense matrices, and design simple and general superfast matrix-vector multiplication algorithms for low-width matrices.

> *We believe that our strongest contribution is in showing that the above classes of matrices, which were seemingly disparate and handled by different specialized algorithms, all fall under the umbrella of low recurrence width and can be handled with* one *class of algorithms.*

Perhaps the poster child for matrices that allow for superfast vector multiplication is the Discrete Fourier Transform. The famous Fast Fourier transform (or FFT) allows matrix-vector multiplication for the Fourier matrix in $O(N\log N)$ operations [15]. Since then numerous followup work have extended this algorithm to solve the matrix vector multiplication algorithm for other, increasingly general structured dense matrices. To motivate our notion of structure, we will focus on the two strands of work that inspired much of our work, and describe how we capture previous results.

Orthogonal polynomial transforms The first strand of work relates to *orthogonal polynomial transforms*. Orthogonal polynomials are widely used and independently important [1–3, 14] - see Section 2 for related work. The transforms can be expressed as matrix-vector multiplication involving matrices $\mathbf{A}$ containing a family of orthogonal polynomials evaluated at points. In particular, the i^{th} row of $\mathbf{A}$ contains the evaluation at points $\{z_1,\dots,z_n\}$ of the i^{th} polynomial $f_i(X)$. Further, these orthogonal polynomials satisfy the following three term recurrence:

$$f_{i+1}(X) = (a_i X + b_i) f_i(X) + c_i f_{i-1}(X),$$

where $a_i, b_i, c_i \in \mathbb{F}$. Driscoll, Healy and Rockmore present an algorithm to perform the orthogonal polynomial transform in $O(N\log^2 N)$ operations [17]. In our first main result, we extended this class of transforms to polynomials that satisfy a more general recurrence.

Definition 1.1. An $N\times N$ matrix $\mathbf{A}$ has recurrence width t if the polynomials $f_i(X) = \sum_{j=0}^{N-1} \mathbf{A}[i,j]X^j$ satisfy

$$f_{i+1}(X) = \sum_{j=0}^{t} g_{i,j}(X) f_{i-j}(X), \tag{1}$$

where the polynomials $g_{i,j} \in \mathbb{F}[X]$ are of degree at most $j+1$.

This is the most basic notion of recurrence width that can be kept in mind as a prototypical example. A more general definition that captures the other strands of work and applications is presented in Definition 3.1.

We note that the orthogonal polynomial recurrence implies that orthogonal polynomial transforms are essentially recurrence width 1 matrices (the exact connection is in Section 10.2). We show in Section 9 that our notion of recurrence width forms a strong hierarchy. In particular, we construct a simple matrix $\mathbf{A}$ with recurrence width $t+1$ such that no matrix with recurrence width t can approximate $\mathbf{Ab}$ for all $\mathbf{b}$ to any reasonable accuracy. Consequently, our extension is a meaningful one; the class of matrices we handle cannot be captured by previous results on orthogonal polynomials. This also justifies our use of the term width in this setting.

Promisingly, our algorithms are optimal in the sense of matching the size of its parametrization, up to poly-log factors.

Theorem 1.2. *Given matrix* $\mathbf{A}$ *with recurrence width* t *and* $\tilde{O}(t^{\omega}N)$ *pre-processing operations (where* ω *is the matrix-matrix multiplication exponent), the product* $\mathbf{Ab}$ *and* $\mathbf{A}^T\mathbf{b}$ *can be found in* $\tilde{O}(t^2N)$ *for any vector* $\mathbf{b}$.

If $\omega = 2$, this is equal to the worst-case input size of a matrix with recurrence width t. Additionally, the recursive structure allows us to generate matrix vector multiplication algorithms for both matrices $\mathbf{A}$ and $\mathbf{A}^T$ in a simple and general way. For example, we recover the bounds of Driscoll et al. [17], and Section 10.2 demonstrates how their algorithm can actually be viewed as a direct application of ours. In certain cases we also get matrix vector multiplication for $\mathbf{A}^{-1}$ and $(\mathbf{A}^{-1})^T$: e.g. for orthogonal polynomials (Section 10.2).

Displacement rank The second relevant strand of work are results for matrices with low *displacement rank*. The notion of displacement rank (which was defined in the seminal work of Kailath et al. [28]) is defined as follows. Given any pair of matrices $(\mathbf{L}, \mathbf{R})$, the displacement rank of $\mathbf{A}$ with respect to $(\mathbf{L}, \mathbf{R})$ is the rank of the *error matrix*:

$$\mathbf{E} = \mathbf{LA} - \mathbf{AR}. \tag{2}$$

To the best of our knowledge, the most powerful results on matrix-vector multiplication for matrices with low displacement rank are in the work of Olshevsky and Shokrollahi [38], who show that any matrix with a displacement rank of r with respect to Jordan form matrices $\mathbf{L}$ and $\mathbf{R}$ can be multiplied with an arbitrary vector with $\tilde{O}(rN)$ operations. (They use these results and matrices to solve the Nevalina-Pick problem as well as solve the interpolation step in some list decoding algorithms in a previous work [37].) Recall that Jordan normal form matrices are special cases of matrices where only the diagonal and superdiagonal can be non-zero. In other words, both $\mathbf{L}$ and $\mathbf{R}$ are 2-*band upper triangular matrices*. In this work we show that when both $\mathbf{L}$ and $\mathbf{R}$ are t-band matrices, any matrix with displacement rank of r with respect to such matrices can be multiplied by a vector with $\tilde{O}((t^2 + tr)N)$ operations.

Theorem 1.3. *Let* $\mathbf{L}$ *and* $\mathbf{R}$ *be triangular* t*-band matrices sharing no eigenvalues, and let* $\mathbf{A}$ *be a matrix such that* $\mathbf{LA} - \mathbf{AR}$ *has rank* r*. Then* $\mathbf{A}$ *and* $\mathbf{A}^T$ *can be multiplied by any vector* $\mathbf{b}$ *in* $\tilde{O}((t^2 + tr)N)$ *operations.*

We note that our results recover the work presented in Olshevsky and Shokrollahi [38], and we believe that our algorithm even for their setting is much simpler. Furthermore, our result can handle more general matrices than band matrices; in general, if one of $\mathbf{L}$ and $\mathbf{R}$ is a band matrix and the other admits superfast multiplication for the associated Krylov matrix.

We find this connection compelling because the set of matrices with low recurrence width and those with low displacement rank seem to be widely different. Indeed the existing algorithms for the class of orthogonal polynomials [17] and low displacement rank [38] look very different. Specifically, the algorithm of Driscoll et al. [17] is a divide and conquer algorithm, while that of Olshevsky and Shokrollahi [38] (and preceding works) heavily exploit structural results on matrices with low displacement ranks. We believe that our strongest conceptual contribution is showing that both of these classes of matrices can be handled with *one* class of algorithms. In particular, it turns out that if one allows for *error polynomials* in recurrences for matrices with recurrence width t then this is enough to handle the case of low displacement rank. In Section 3 we present the more abstract recurrence that captures both of these classes of matrices. More importantly, we present efficient algorithms that work for these general recurrences. We believe that unifying these existing threads of disparate work is interesting in its own right.

As we have pointed out earlier, orthogonal polynomials and low displacement rank matrices have applications in numerous areas from signal processing to machine learning. Indeed orthogonal polynomials have their own dedicated conference [4]. For more details on matrices with low displacement rank as well as its application, please refer to the survey by Kailath and Sayed [29]. Our matrices naturally inherit these applications.

1.2 Our Algorithm and Techniques

At the heart of our basic algorithms is (in hindsight the simple) observation that given a matrix with recurrence width t, one can re-initialize the recurrence at any point as long as we know the values of t consecutive rows of the matrix. By pre-computing such rows selectively, we are able to break the problem into two independent halves and design a divide and conquer algorithm.

Throughout this paper it is helpful to keep the following prototypical example in mind: Consider a matrix $\mathbf{A} \in \mathbb{F}^{N\times N}$ with associated polynomials $f_i(X) = \sum_{j=0}^{N-1} \mathbf{A}[i,j]X^j$ that satisfy a recurrence (1). Assume that the $g_{i,j}(X)$ are polynomials satisfying $\deg g_{i,j}(X) \le j+1$ and that $\deg f_i(X) \le i$. Note that the orthogonal polynomial transforms already fall under this limited case.

Basic algorithms The simplest algorithm is for computing $\mathbf{A}^T\mathbf{b}$ for any vector $\mathbf{b}$. Because of the correspondence between vectors and polynomials, this amounts to computing the combination $\sum \mathbf{b}[i] f_i(X)$. By the recurrence, each $f_i(X)$ can be written as a linear combination of the $f_0(X),\dots,f_t(X)$, where the coefficients of the combination are polynomials of degree approximately i (this is formalized in Lemma 3.5). However, for $i \ge N/2$, this can be broken down into two parts: the linear combination of $f_i(X)$ in terms of $f_{N/2}(X),\dots,f_{N/2+t}(X)$, and the dependence of $f_{N/2}(X),\dots,f_{N/2+t}(X)$ on $f_0(X),\dots,f_t(X)$. The former has size approximately $i-N/2$, and the latter $N/2$. So that the sum $\sum \mathbf{b}[i] f_i(X)$ is broken into two parts for $i < N/2$ and $i \ge N/2$, and the latter is re-initialized as a recurrence starting from $N/2$. Thus the common theme is: break the recurrence into two recurrences of half the size, the first half initialized from 0, the second half initialized from $N/2$, and the "jump" from f_0 to $f_{N/2}$.

This idea is illustrated particularly simply when $t=0$, so that $f_i(X) = \prod_{j=0}^{i-1} g_{j,0}(X)$. In this case, the quantity to compute becomes

$$\sum_{i=0}^{N-1} \mathbf{b}[i] f_i(X) = \left(\sum_{i=0}^{N/2-1} \mathbf{b}[i] g_{0,0}(X)\cdots g_{i-1,0}(X) \right) + f_{N/2}(X) \left(\sum_{i=N/2}^{N-1} \mathbf{b}[i] g_{N/2,0}(X)\cdots g_{i-1,0}(X) \right)$$

As shown, the whole sum is split into two smaller sums of the same form, provided we can compute $f_{N/2}(X) = g_{0,0}(X)\cdots g_{N/2-1,0}(X)$. Thus the entire sum can be computed in quasi-linear time if we know $\prod_{i=kN/2^d}^{(k+1)N/2^d-1} g_{i,0}(X)$ for all $d \in [\log N]$, $k \in [2^d]$. These can be pre-computed because they depend only on $\mathbf{A}$ and not $\mathbf{b}$.

Computing the product $\mathbf{Ab}$ is more difficult because each polynomial $f_i(X)$ is not treated as a "single entity" as in $\mathbf{A}^T\mathbf{b}$. Rather, the coefficients are individual manipulated - in particular, we need to compute the dot product of the coefficient vector of $f_i(X)$ with $\mathbf{b}$. Here we use the observation that the *dot product* between two vectors is a specific element of their *convolution*. Thus if $b(X) = \sum \mathbf{b}[i]X^{N-i}$, the entries of $\mathbf{Ab}$ are the coefficients of X^{N-1} in $b(X), g_{0,0}(X)b(X),\dots,g_{0,0}(X)\dots g_{N-1,0}(X)b(X)$. The second half of this is the coefficients of X^{N-1} in $f_{N/2}(X)b(X),\dots,f_{N/2}(X)g_{N/2,0}(X)\dots g_{N-1,0}(X)b(X)$ which becomes a similar problem of half the size by defining $b'(X) = f_{N/2}(X)b(X)$. The details of this algorithm are in Section 6.

Extensions of the Recurrence Many of our applications require considering more complicated recurrences than (1). One of our main new contributions, which has allowed us to reduce many problems including displacement rank to the notion of recurrence width, is in identifying and solving these extensions]? The main generalizations we consider are

1. We allow for the rows of $\mathbf{A}$ satisfy (1) with some error terms. That is, the recurrence takes the form

$$f_{i+1}(X) = \sum_{j=0}^{t} g_{i,j}(X) f_{i-j}(X) + E_{i+1}(X) \tag{3}$$

   where the matrix $\mathbf{E}$ formed by putting the coefficients of $E_i(X)$ on its rows has low rank.

2. Consider a rational recurrence modulo some polynomial

$$D_{i+1}(X)\cdot f_{i+1}(X) \equiv \sum_{j=0}^{t} g_{i,j}(X) f_{i-j}(X) \pmod{M(X)} \tag{4}$$

   where for each i, $\gcd(D_i(X), M(X)) = 1$ and we define $f_{i+1}(X)$ to be the unique polynomial of the smallest degree that satisfies the above equation.

   An equivalent way of thinking about this is considering the recurrence coefficients $g_{i,j}(X)$ in (1) to lie in the quotient ring $\mathbb{F}[X]/(M(X))$ instead of $\mathbb{F}[X]$.

3. We replace the variable X with a matrix $\mathbf{R} \in \mathbb{F}^{N\times N}$, such that

$$\mathbf{f}_{i+1} \equiv \sum_{j=0}^{t} g_{i,j}(\mathbf{R})\mathbf{f}_{i-j} \tag{5}$$

and $\mathbf{f}_i = \mathbf{A}[i,:]$ is directly interpreted as a vector instead of polynomial. This essentially recovers (1) when $\mathbf{R} = \mathbf{S}$, the shift matrix.

Our connection to multipoint evaluation of multvariate polynomials involves matrices that have the form of both (3) and (4), and the solution to (5) itself reduces to a problem of that form. Of particular interest is that in Section 10.1, we show how the displacement rank equation (2) (where $\mathbf{L}, \mathbf{R}$ are triangular t-banded matrices) allows us to write the rows of $\mathbf{A}$ in the form of a recurrence using all three of these extensions:

$$D_{i+1}(X)\cdot \mathbf{f}_{i+1} \equiv \left(\sum_{j=0}^{t} g_{i,j}(\mathbf{R})\mathbf{f}_{i-j}\right) + \mathbf{E}[i+1,:] \pmod{c_{\mathbf{R}}(X)}. \tag{6}$$

These first two extensions can be reduced fairly directly to the basic recurrence (1); this is done in Section 6.2 and Section 7 respectively. We elaborate on the third extension because it involves an intermediate problem with potentially broader uses.

Matrix Functions and Krylov Efficiency Recurrence (5) involves evaluating functions at a matrix $\mathbf{R}$; we focus on triangular t-banded matrices which is sufficient for displacement rank. The question of evaluating a function of a matrix is itself a well-studied and useful problem [26]. Classical ways of computing these matrix functions use natural decompositions of the matrix, such as its eigendecomposition or Jordan normal form $\mathbf{R} = \mathbf{A}\mathbf{J}\mathbf{A}^{-1}$. The evaluation of an analytic function then becomes $f(\mathbf{R}) = \mathbf{A}f(\mathbf{J})\mathbf{A}^{-1}$ which admits superfast multiplication if each component does. We note that the Jordan decomposition is generally hard to compute and much work has been done on computing specific matrix functions without going through the Jordan form [41]. Despite this, in Section 10.5 we show that it is possible to compute the Jordan decomposition quickly for several special subclasses of the matrices we are interested in, using techniques involving low-width recurrences.

However, solving (5) has more structure and is easier than evaluating general matrix functions. Consider again the simplified case when $t = 0$. Fixing the $g_{i,j}(X)$, we can define the polynomials $f_i(X)$ by (1). Note that $\mathbf{f}_i = g_{i-1,0}(\mathbf{R})\cdots g_{0,0}(\mathbf{R})\mathbf{f}_0 = f_i(\mathbf{R})\mathbf{f}_0$. So we can factor $\mathbf{A} = \mathbf{A}'\mathbf{K}$, where $\mathbf{A}'$ is the coefficient matrix of the basic polynomial recurrence (Definition 1.1), and $\mathbf{K}$ is the matrix whose i-th column is $\mathbf{R}^i\mathbf{f}_0$. Thus this reduces to the basic recurrence (1) if we can multiply by $\mathbf{K}$, the *Krylov matrix* on $\mathbf{R}$ and $\mathbf{f}_0$.[1] We say that $\mathbf{R}$ is *Krylov efficient* if all of its Krylov matrices admit superfast multiplication.

In Section 8, we show that the Krylov matrix itself satisfies a recurrence of type (3),(4) of width t. Using our established results, this gives a single $\tilde{O}(t^2N)$ algorithm that unifies the previous subclasses with Jordan decompositions, and implies all triangular banded matrices are Krylov efficient.

We remark that the Krylov efficiency concept does not apply only to our problem. If $\mathbf{K}$ is the Krylov matrix on $\mathbf{A}$ and $\mathbf{b}$, then $\mathbf{K}\mathbf{b} = \sum \mathbf{b}[i]\mathbf{A}^i\mathbf{x}$ is naturally related to contexts involving Krylov subspaces, matrix polynomials, and so on. The product $\mathbf{K}^T\mathbf{b} = [\mathbf{b}\cdot\mathbf{x}, \mathbf{b}\cdot\mathbf{A}\mathbf{x}, \mathbf{b}\cdot\mathbf{A}^2\mathbf{x}, \ldots]$ is also useful; it is the first step in the Wiedemann algorithm for computing the minimal polynomial or kernel vectors of a matrix $\mathbf{A}$ [30].

Quick comparison with existing work Finally, we again highlight that we consider our techniques to be relatively simple, but they work just as well as more involved techniques that only work in specific cases. In many of the past cases, the algorithms for $\mathbf{Ab}$ and $\mathbf{A}^T\mathbf{b}$ would require different techniques, but in our case we get both the results for a large class of structured dense matrices with essentially the same idea. Further, some of the existing results that are based in structural results (e.g. those based on the Bezoutian [22] and on displacement rank [38]) seems to invoke some fairly sophisticated algebraic techniques while our results use mostly combinatorial techniques (in particular divide and conquer) except for polynomial interpolation and evaluation. Again our simpler techniques give results that work for matrices that cannot be handled with these previous works.

[1]The image of this matrix is the Krylov subspace on $\mathbf{R}$ and $\mathbf{f}_0$.

Because of their simplicity, we believe that these algorithms are also practical and should be competitive against existing algorithms that have been implemented. Our preliminary experimental results, presented in Section 10.7, have been encouraging and we plan to explore more rigorous experimental results in future work.

1.3 Some Consequences and More Context

We believe that given the fundamental nature of the matrix vector multiplication problem, our main contribution is presenting the notion of recurrence width for matrices for which we give optimal algorithms (up to poly-log factors) and in the process unify quite a few known results. However, our results have consequences beyond just the results in matrix-vector multiplication. We collect some of our favorite ones here.

A natural question to ask is if the matrices with recurrence width t (for say non-constant t) contain any interesting class of matrices beyond the fact that (i) these contain both the classes of matrices considered in [17] and [38] and (ii) these matrices have nice algorithmic properties. We first present a few examples of matrices that have been studied before that indeed fall in this general category of matrices. In particular, we present applications in the following order: (1) We present matrices that arise naturally in coding theory and (2) We present matrices that arise when computing some well known sequences. Further, we would like to point out that our generic algorithms solve problems from these different areas each of which typically have their own home-grown algorithms and in some cases (e.g. Bernoulli numbers) we almost match the best known runtimes of these more specific algorithms.

Matrices from Coding Theory. We first observe that the problem of multipoint evaluation of multivariate polynomials (i.e. given a multivariate polynomial $f(X_1,\dots,X_m)$ over $\mathbb{F}$ and N evaluation points $\mathbf{a}_1,\dots,\mathbf{a}_N \in \mathbb{F}^m$ output the vector $(f(\mathbf{a}_i))_{i\in[N]}$) corresponds to matrix-vector multiplication for matrices with recurrence width t (where t is polynomial but sub-linear in N). Further, the encoding problem of recently introduced multiplicity codes (which have excellent local decoding properties [32–34]), which in turn corresponds to evaluating multivariate polynomials and all of their derivatives up to a certain order, again corresponds to matrix vector multiplication with a matrix with recurrence width t (where t is polynomial but sub-linear in N). This was already observed in the context of list decoding Reed-Solomon codes for the case of $m = 2$ by Olshevsky and Shokrollahi [37] (we extend this observation to $m > 2$). While we do not prove any new upper bounds for these probelms, it turns out that the connection to multipoint evaluation of multivariate polynomials has some interesting complexity implications for our result. In particular, recall that the worst-case input size of a matrix with recurrence width t is $\Theta(t^2N)$ (and our algorithms are optimal with respect to this input measure). However, it is natural to wonder if one can have faster algorithms with respect to a more per-instance input size. A natural such case would to be to represent the coefficients $g_{i,j}(X)$ in (1) as *sparse* polynomials and count the input size as the total number of non-zero coefficients in all the $g_{i,j}(X)$'s. Another natural representation would be for cases where we can guarantee $\deg(g_{i,j}(X)) \le d < t$: in such a case can we design efficient algorithms with respect to the input size $\Theta(tdN)$? In both cases we show that improving our generic algorithm substantially will improve the state of the art results in multipoint evaluation of multivariate polynomials over arbitrary fields.[2] We present the details in Section 11.

Computing sequences. Second we observe that the recurrences that we consider for our matrices actually have *holonomic* sequences as a special case, which are some of the very well studied sequences and many algorithms for such sequences have been implemented in algebra packages [44]. Additionally, the computation of many well-known sequences of numbers, including Stirling numbers of the second kind and Bernoulli numbers, are also very well studied problems. There have been some recent ad-hoc algorithms to compute such numbers (e.g. the current best algorithm to compute the Bernoulli numbers appears in [25]). Despite these sequences not being holonomic, our general purpose algorithms can still be to compute them. In Section 10.4, we show how to compute the first N Bernoulli numbers in $O(N\log^2 N)$ operations, which recovers the same algorithmic bit complexity (potentially with a log factor loss) as the algorithms that are specific to Bernoulli numbers.

[2]We note that for *finite* fields, Kedlaya and Umans [31] have essentially solved this problem. The case for general fields however, is much less explored and (somewhat to our surprise) widely open.

2 Related Work

Superfast structured matrix vector multiplication has been a rich area of research. Toeplitz, Hankel, and Vandermonde matrices and their inverses all have classical superfast multiplication algorithms that correspond to operations on polynomials [7, 10]. A superfast algorithm was developed for Cauchy matrices (and slight generalizations) was developed by Gerasoulis in 1988 [20]. Multiplication with Cauchy matrices corresponds to operations on rational polynomials; Cauchy matrices naturally fit in with the other three types of matrices. The four classes of matrices are all generalized by the notion of displacement rank introduced by Kailath, Kung, and Morf in 1979 [28]. Kailath et al. used displacement rank to define Toeplitz-like matrices, generalizing Toeplitz matrices. In 1994, Gohberg and Olshevsky further used displacement rank to define Vandermonde-like, Hankel-like, and Cauchy-like matrices and developed superfast algorithms for vector multiplication [22]. These matrix classes were unified and generalized by Olshevsky and Shokrollahi in 2000 [38] with a class of matrices they named confluent Cauchy-like. Confluent Cauchy-like matrices are those with low displacement rank with respect to Jordan form matrices; we extend these results by investigating matrices with low displacement rank with respect to any triangular Δ-band matrices, which we define to be matrices whose non-zero elements all appear in Δ consecutive diagonals. Algorithms for these four classes of matrices have continued to be refined for precision; we refer to Pan and Tsigaridas in 2014 for an overview and for an algorithm with explicit bounds on precision [40].

Our work is spiritually closer to the study of orthogonal polynomial transforms, especially that of Driscoll, Healy, and Rockmore [17]. Orthogonal polynomials are widely used and well worth studying in their own right: for an introduction to the area, see the classic book of Chihara [14]. We present applications for some specific orthogonal polynomials. Chebyshev polynomials are used for numerical stability (see e.g. the ChebFun package [5]) as well as approximation theory (see e.g. Chebyshev approximation [1]). Jacobi polynomials form solutions of certain differential equations [2]. Zernike polynomials have applications in optics and physics [3]. In fact, our investigation into structured matrix-vector multiplication problems started with some applied work on Zernike polynomials, and our results applied to fast Zernike transforms have been used in improved cancer imaging [43]. Driscoll et al. rely heavily on the three-term recurrence satisfied by orthogonal polynomials to devise a divide-and-conquer algorithm for computing matrix-vector multiplication. Our first main result is a direct generalization of the recurrence, and we rely heavily on the recurrence to formulate our own divide and conquer algorithm. As discussed earlier, we view the connection of these two strands of work as our strongest conceptual contribution.

A third significant strand of research is the study of semiseparable matrices. We refer to an extensive survey by Vandebril, Van Barel, Golub, and Mastronardi [42] for a detailed discussion of the body of work, but we provide a brief commentary here. The most straightforward class of semiseparable matrices are the generator representable semiseparable matrices, which are matrices whose upper triangular and lower triangular portions are both of (low) rank. Our results can straightforwardly recover generator semiseparable matrices whose triangular portions are of rank 1. However, semiseparable matrices are defined more generally with respect to the rank of matrix sub-blocks. The idea of utilizing the ranks of matrix sub-blocks has been generalized many times, and we refer to Bella, Eidelman, Gohberg, and Olshevsky's work on (H, m)-quasiseparable matrices [45] for a relatively recent exploration of various generalizations. This generalization actually has deep connections to polynomials that satisfy recurrences; if we define $p_i(X)$ as the characteristic polynomial of the upper left $i \times i$ submatrix, the p_i form a family that satisfy recurrence relations. Connecting our notion of recurrence - where the rows of our matrix form a recursive polynomial family - to that of quasiseparable matrices will be explored in future work. As far as we are aware, there is no fully general superfast matrix-vector multiplication algorithm for (H, m)-quasiseparable matrices. We note that this is an area of great active research; we point the reader to Bella et al.'s survey on computing with quasiseparable matrices [9] for an overview of the wide array of work.

3 Preliminaries

3.1 Notation

We will use $\mathbb{F}$ to denote a field and use $\mathbb{R}$ and $\mathbb{C}$ to denote the field of real and complex numbers respectively.[3][4] The set of polynomials over $\mathbb{F}$ and set of rational functions over $\mathbb{F}$ will be denoted by $\mathbb{F}[X]$ and $\mathbb{F}(X)$ respectively. For polynomials $p(X), q(X) \in \mathbb{F}[X]$, we use the notation $p(X) \equiv q(X) \pmod{M(X)}$ to indicate equivalence modulo $M(X)$, i.e. $M(X)|(p(X) - q(X))$, and $p(X) = q(X) \pmod{M(X)}$ to specify $p(X)$ as the unique element of this equivalence class with degree less than $\deg M(X)$. We will sometimes consider polynomials over multiple indeterminates; if $f(X_1, \ldots, X_k) \in \mathbb{F}[X_1, \ldots, X_k]$, we use $\deg(f)$ to mean the maximum degree of its monomials (i.e. sum of powers of all indeterminates), and $\deg_{X_i}(f)$ to denote the degree of f when viewed as a polynomial over X_i. For any integer $m \ge 1$, we will use $[m]$ to denote the set $\{1, \ldots, m\}$. Unless specified otherwise, indices in the paper start from 0.

In this paper, vectors are boldset like $\mathbf{x}$ and are column vectors. Unless specifically mentioned otherwise we will assume that $\mathbf{x} \in \mathbb{F}^N$. We will denote the ith element in $\mathbf{x}$ by $\mathbf{x}[i]$ and the vector between the positions $[\ell, r) : \ell \le r$ by $\mathbf{x}[\ell : r]$. For any subset $T \subseteq [N]$, $\mathbf{e}_T$ denoted the characteristic vector of T. We will shorten $\mathbf{e}_{\{i\}}$ by $\mathbf{e}_i$.

Matrices will be boldset like $\mathbf{M}$ and by default $\mathbf{M} \in \mathbb{F}^{N \times N}$. We will denote the element in the ith row and jth column by $\mathbf{M}[i, j]$. $\mathbf{M}[\ell_1 : r_1, \ell_2 : r_2]$ denotes the sub-matrix $\{\mathbf{M}[i, j]\}_{\ell_1 \le i < r_1, \ell_2 \le j < r_2}$. In particular we will use $\mathbf{M}[i, :]$ and $\mathbf{M}[:, j]$ to denote the ith row and jth column of $\mathbf{M}$ respectively. ($\mathbf{M}[0, 0]$ denotes the 'top-left' element of $\mathbf{M}$.) We will use $\mathbf{S}$ to denote the shift matrix (i.e. $\mathbf{S}[i, j] = 1$ if $j = i + 1$ and 0 otherwise) and $\mathbf{I}$ to denote the identity matrix. We will use $\mathbf{V}$ to denote the Vandermonde matrix. In particular if the 'evaluation points' are $\alpha_0, \ldots, \alpha_{N-1}$, then $\mathbf{V}[i, j] = \alpha_i^j$. Given a matrix $\mathbf{A}$, we denote its transpose and inverse (assuming it exists) by $\mathbf{A}^T$ (so that $\mathbf{A}^T[i, j] = \mathbf{A}[j, i]$) and $\mathbf{A}^{-1}$ (so that $\mathbf{A} \cdot \mathbf{A}^{-1} = \mathbf{I}$). We will denote $(\mathbf{A}^T)^{-1}$ by $\mathbf{A}^{-T}$.

Given a matrix $\mathbf{M} \in \mathbb{F}^{N \times N}$ and a vector $\mathbf{b} \in \mathbb{F}^N$, the *Krylov matrix of* $\mathbf{M}$ *generated by* $\mathbf{b}$ (denoted by $\mathcal{K}(\mathbf{M}, \mathbf{b})$) is the $N \times N$ matrix whose ith column for $0 \le i < N$ is given by $\mathbf{M}^i \cdot \mathbf{b}$. We say that $\mathbf{M}$ is (α, β)-*Krylov efficient* if for every $\mathbf{b} \in \mathbb{F}^N$, we have that $\mathbf{K} = \mathcal{K}(\mathbf{M}, \mathbf{b})$ admits the operations $\mathbf{Kx}$ and $\mathbf{K}^T\mathbf{x}$ (for any $\mathbf{x} \in \mathbb{F}^N$) with $\tilde{O}(\beta N)$ many operations (with $\tilde{O}(\alpha N)$ pre-processing operations). (The $\tilde{O}(\cdot)$ notation is defined below.) Section 8 has examples of Krylov efficient matrices.

Finally, we address some issues related to runtime analyses. We will use $\tilde{O}(T(N))$ to denote $O\left(T(N) \cdot \log^{O(1)}(T(N))\right)$. We will denote the *size* of a polynomial $p(X) \in \mathbb{F}[X]$ as the degree of $p(X)$. The size of a fraction $g(X) = \frac{a(X)}{b(X)}$, denoted by $\|g\|$ is the sum of sizes of $a(X)$ and $b(X)$. We will call a set of fractions $S \subseteq \mathbb{F}(X)$ to be *nice* if for any $g_1(X), g_2(X) \in S$, we have

1. $\|g_1 + g_2\| \le \max(\|g_1\|, \|g_2\|)$. Further $g_1(X) + g_2(X)$ can be computed in $\tilde{O}(\|g_1 + g_2\|)$ operations over $\mathbb{F}$.

2. $\|g_1 \cdot g_2\| \le \|g_1\| + \|g_2\|$. Further $g_1(X) \cdot g_2(X)$ can be computed in $\tilde{O}(\|g_1 \cdot g_2\|)$ operations over $\mathbb{F}$.

We note that any subset of $\mathbb{F}(X)$, where all the fractions have the same denominator (e.g. $\mathbb{F}[X]$), is nice.

3.2 Known Results

Multiplication of two degree N univariate polynomials can be performed in $\tilde{O}(N)$ operations: The classic FFT computes it in $O(N \log N)$ operations for certain fields [15], and generalizations of the Schonhage-Strassen algorithm compute it in $O(N \log N \log\log N)$ ring operations in general [12]. Computing polynomial divisors and remainders, i.e. $p(X) \pmod{q(X)}$ with $\deg(p(X)), \deg(q(X)) = O(N)$ can be done with the same number of operations [13].

The matrix-vector product by matrices $\mathbf{A}, \mathbf{A}^T$ (and $\mathbf{A}^{-1}, \mathbf{A}^{-T}$ when they exist) for Toeplitz and Hankel matrices $\mathbf{A}$ can be computed in $O(N \log N)$ operations [39]. When $\mathbf{A}$ is a Vandermonde matrix, these products takes $O(N \log^2 N)$ operations [39].

The characteristic polynomial $c_{\mathbf{M}}(X)$ of a matrix $\mathbf{M}$ is equal to the determinant of $X\mathbf{I} - \mathbf{M}$. When $\mathbf{M}$ is triangular, it is equal to $\prod(X - \mathbf{M}[i, i])$. By the Cayley-Hamilton Theorem, every matrix satisfies its own characteristic equation, i.e. $c_{\mathbf{M}}(\mathbf{M}) = 0$.

[3] Assume that $\mathbb{F}$ is large enough and supports the FFT. Otherwise, the runtime bounds occur an extra $\log\log N$ factor.

[4] All results in this paper hold if $\mathbb{F}$ is replaced by a commutative ring. We use fields for clarity; our known applications use $\mathbb{R}$ or $\mathbb{C}$.

A λ-*Jordan block* is a square matrix of the form $\lambda\mathbf{I}+\mathbf{S}$, where $\mathbf{S}$ is the shift matrix which is 1 on the superdiagonal and 0 elsewhere. A matrix in *Jordan normal form* is a direct sum of Jordan blocks. The minimal polynomial of a matrix is equal to the product $\prod_\lambda (X-\lambda)^{n_\lambda}$ where n_λ is the size of the largest Jordan block for λ. Consequently, if a matrix has equal minimal and characteristic polynomials, then it has only one Jordan block per eigenvalue.

3.3 Our Problem

In this paper, we are interested in the matrix-vector multiplication problem. In other words, given $\mathbf{A} \in \mathbb{F}^{N\times N}$ and $\mathbf{b} \in \mathbb{F}^N$, we want to compute $\mathbf{c} = \mathbf{Ab}$ such that for every $0 \le i < N$:

$$\mathbf{c}[i] = \sum_{j=0}^{N-1} \mathbf{A}[i,j]\cdot\mathbf{b}[j].$$

For the rest of the paper we will assume that $N = 2^m$: this does not change the asymptotics but makes some of the subsequent notations simpler. We will be interested in matrices $\mathbf{A}$ for which we can compute $\mathbf{Ab}$ and $\mathbf{A}^T\mathbf{b}$ efficiently. In particular, we will be interested in structured matrices $\mathbf{A}$ such that its rows satisfy certain recurrences. For notational convenience, define $\mathbf{f}_i^T = \mathbf{A}[i,:]$.

Definition 3.1. Let $\mathscr{R}$ be a ring and $\otimes : \mathscr{R}\times\mathbb{F}^N \to \mathbb{F}^N$ be an operator satisfying

$$a\otimes(b\otimes\mathbf{z}) = (a\cdot b)\otimes\mathbf{z} \tag{7}$$

$$a\otimes\mathbf{z} + b\otimes\mathbf{z} = (a+b)\otimes\mathbf{z} \tag{8}$$

$$1\otimes\mathbf{z} = \mathbf{z} \tag{9}$$

A matrix $\mathbf{A}\in\mathbb{F}^{N\times N}$ has recurrence width t if and only if its rows $\mathbf{f}_i^T = \mathbf{A}[i,:]$ satisfy

$$\mathbf{f}_{i+1} = \sum_{j=0}^{t} g_{i,j}\otimes\mathbf{f}_{i-j} \tag{10}$$

for $i > t$.

The $\otimes$ operator provides an abstraction so that the standard form (10) captures the recurrence variants and extensions we are interested in. We note that the $\otimes$ operator essentially induces a left $\mathscr{R}$-module structure over $\mathbf{F}^N$ [18]. We focus on when $\mathscr{R}$ is a subring or quotient ring of $\mathbb{F}(X)$ so that each $g_{i,j}$ can be represented by an element $g_{i,j}(X)\in\mathbb{F}(X)$. We additionally assume that $\|g_{i,j}(X)\| \le j+1$. Finally, we call a recurrence in (10) to be *nice* if all set of all coefficients (and its closure) is nice.

The basic polynomial recurrence as in Definition 1.1 as well as its extensions (aside from the recurrence with error (3)) can all be viewed as various instantiations of the $\otimes$ operator.

Definition 3.2. Let $\mathscr{R} = \mathbb{F}[X]$ and define $a(X)\otimes\mathbf{z}$ to be the convolution between the coefficient vector of a and $\mathbf{z}$.

This defines recurrence (1) and is the main type of recurrence we consider. The classic case is when $\deg(f_i)\le i$ for $0\le i\le t$, so that $\deg(g_{i,j})\le j+1$ implies that $\deg(f_i)\le i$ for all i (which is true for orthogonal polynomials with $t=1$). When the former does not hold, it may happen that $\deg(f_i)\ge N$, but we can still define the matrix $\mathbf{A}$ by cutting off the higher order terms, i.e. let $\mathbf{A}[i,j]$ be the coefficient of X^j in $f_i(X)$. This is more precisely an instance of the next definition of $\otimes$ with $M(X) = X^N$.

Definition 3.3. Let $\mathscr{R} = \mathbb{F}[X]/(M(X))$ for $M(X)\in\mathbb{F}[X]$ of degree N, and define $a(X)\otimes\mathbf{z}$ as in Definition 3.2. More precisely, it is the coefficient vector of $b(X)$ where $b(X) = a(X)\cdot(\sum\mathbf{z}[i]X^i) \pmod{M(X)}$.

Note that Definition 3.3 captures recurrence (4): that equation is equivalent to writing $\mathbf{f}_i = \sum(g_{i,j}/D_{i+1})\otimes\mathbf{f}_{i-j}$ using this definition of $\otimes$.

Definition 3.4. Let $\mathbf{R}\in\mathbb{F}^{N\times N}$ and $\mathscr{R} = \mathbb{F}[X]/(M(X))$ for a degree N polynomial $M(X)$. Given $a(X)\in\mathscr{R}$, let $a(X)\otimes\mathbf{z} = a(\mathbf{R})\mathbf{z}$.

As usual the prototypical *polynomial recurrence* is through Definition 3.2. The other cases will reduce to this: Definition 3.3 is covered in Section 6.2 and Definition 3.4 in Section 6.2. We call a **R**-*matrix recurrence* one that is defined by Definition 3.4.

Finally, we will use $\mathbf{G} \in (\mathbb{F}(X))^{N\times(t+1)}$ to compactly represent the input: we will set $\mathbf{G}[i,j] = g_{i,j}(X)$ for $0 \le i < N$ and $0 \le j \le t$. Further, we will use $\mathbf{F} \in (\mathbb{F}(X))^{t+1}$ to contain the initial condition, i.e. $\mathbf{F}[i] = \mathbf{f}_i$ for $0 \le i \le t$. A common instantiation of this vector will be when $\mathbf{f}_i = \delta_{i,j}$ for some $0 \le j \le t$. In this case we will denote $\mathbf{F}$ by $\mathbf{e}_j$. Note that the pair $(\mathbf{G},\mathbf{F})$ completely specifies the recurrence and we will refer to it simply as $(\mathbf{G},\mathbf{F})$-recurrence.

3.4 Examples of matrices with low recurrence width

For concreteness, we provide a few typical cases of matrices with low recurrence width and show how they fall under the above definitions.

Orthogonal Polynomials The matrix

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ -1 & 0 & 2 & 0 & 0 \\ 0 & -3 & 0 & 4 & 0 \\ 1 & 0 & -8 & 0 & 8 \end{bmatrix}$$

has recurrence width 1 under Definition 3.2 with $g_{i,0} = 2x$, $g_{i,1} = -1$ for all i. This corresponds to Chebyshev polynomials of the first kind; the matrix $\mathbf{A}$ encodes their coefficients.

Now consider

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 2 & 3 & 4 \\ -1 & 1 & 7 & 17 & 31 \\ 0 & 1 & 26 & 99 & 244 \\ 1 & 1 & 97 & 577 & 1921 \end{bmatrix}$$

This matrix also satisfies (10) with the same $g_{i,j}$ as above, but where $\otimes$ is defined by 3.4 with $\mathbf{R} = \operatorname{diag}(0,1,2,3,4)$. For example, $\mathbf{f}_0, \mathbf{f}_1, \mathbf{f}_2$ satisfy the relation

$$\begin{bmatrix} -1 \\ 1 \\ 7 \\ 17 \\ 31 \end{bmatrix} = 2\mathbf{R} \begin{bmatrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} - \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

Note in particular that if $f_i(X)$ is the ith Chebyshev polynomial, then $\mathbf{A}[i,j] = f_i(j)$. Thus this matrix is actually the orthogonal polynomial transform of Chebyshev polynomials at points $\{0,1,2,3,4\}$.

Displacement Rank Toeplitz, Vandermonde, and Cauchy matrices all have very simple displacement rank structure. For a slightly more complex example, we examine the Pascal matrix

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 \\ 1 & 3 & 6 & 10 \\ 1 & 4 & 10 & 20 \end{bmatrix},$$

which does not fall under the above categories. By Pascal's identity $\mathbf{A}[i,j] = \mathbf{A}[i-1,j] + \mathbf{A}[i,j-1]$, so this matrix satisfies

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \mathbf{A} - \mathbf{A} \begin{bmatrix} 1 & -1 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

so that it has displacement rank 1 with respect to Jordan matrices $\mathbf{L}, \mathbf{R}$, so it is a confluent Cauchy-like matrix [38]. Isolating the $i+1$th row yields $\mathbf{f}_i^T - \mathbf{f}_{i+1}^T \mathbf{R} = \mathbf{0}^T$, so $\mathbf{f}_{i+1} = \mathbf{R}^{-T}\mathbf{f}_i$. In Appendix A, we show how this can be viewed as an instance of (10) under Definition 3.4.

3.5 The work-horse lemma

Next, we present a lemma that formalizes the simple notion that one can re-initialize the recurrence from any $\mathbf{f}_k, \mathbf{f}_{k+1}, \dots, \mathbf{f}_{k+t}$ (where k need not be 0). This notation will be useful throughout the main algorithms.

First, for any $0 \le i < n$, we will define the $(t+1)\times(t+1)$ matrix:

$$\mathbf{T}_i = \begin{pmatrix} g_{i,0}(X) & g_{i,1}(X) & \cdots & g_{i,t-1}(X) & g_{i,t}(X) \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \dots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \end{pmatrix}.$$

And for any $\ell \le r$, define

$$\mathbf{T}_{[\ell:r]} = \mathbf{T}_{r-1} \times \cdots \times \mathbf{T}_{\ell}.$$

Conceptually, this lemma states that every term in the recurrence can be written as a combination of any $t+1$ consecutive terms and provides properties about these combination coefficients. It is helpful to consider the polynomial recurrence case (1), whence every term is a linear combination (with polynomial coefficients) of previous consecutive terms.

Lemma 3.5. *Given a sequence $\mathbf{f}_0, \dots, \mathbf{f}_N$ specified by (10), for every index k, there exist $h_{i,j}^{(k)}(X) \in \mathbb{F}(X)$ for $0 \le i < N-k, 0 \le j \le t$ such that*

1. $\mathbf{f}_{i+k} = \sum_{j=0}^{t} h_{i,j}^{(k)}(X) \otimes \mathbf{f}_{k+j}$
2. $h_{i+1,j}^{(k)}(X) = \sum_{\ell=0}^{t} g_{i+k,\ell}(X) h_{i-\ell,j}^{(k)}(X)$ *for* $i \ge t$ *and* $h_{i,j}(X) = \delta_{i,j}$ *for* $i \le t$.

Proof. Note that (2) can be written as

$$\mathbf{f}_{i+1} = \begin{bmatrix} g_{i,0}(X) & \cdots & g_{i,t}(X) \end{bmatrix} \otimes \begin{bmatrix} \mathbf{f}_i & \cdots & \mathbf{f}_{i-t} \end{bmatrix}^T$$

or

$$\begin{bmatrix} \mathbf{f}_{i+1} & \cdots & \mathbf{f}_{i-t+1} \end{bmatrix}^T = \mathbf{T}_i \otimes \begin{bmatrix} \mathbf{f}_i & \cdots & \mathbf{f}_{i-t} \end{bmatrix}^T$$

By composing this, we get

$$\begin{aligned} \begin{bmatrix} \mathbf{f}_{k+i} & \cdots & \mathbf{f}_{k+i-t} \end{bmatrix}^T &= \mathbf{T}_{k+i-1} \otimes \left(\cdots \otimes \left(\mathbf{T}_k \otimes \begin{bmatrix} \mathbf{f}_{k+t} & \cdots & \mathbf{f}_k \end{bmatrix}^T \right) \right) \\ &= (\mathbf{T}_{k+i-1} \cdot \mathbf{T}_{k+i-2} \cdots \mathbf{T}_k) \otimes \begin{bmatrix} \mathbf{f}_{k+t} & \cdots & \mathbf{f}_k \end{bmatrix}^T \\ &= \mathbf{T}_{[k:k+i]} \otimes \begin{bmatrix} \mathbf{f}_{k+t} & \cdots & \mathbf{f}_k \end{bmatrix}^T \end{aligned}$$

The first part of the lemma is equivalent to the top row of this equation, where we define $h_{i,j}^{(k)}$ to be $(1,j)$-th entry of $\mathbf{T}_{[k:k+i]}$. The second part of the lemma is equivalent to the top row of $\mathbf{T}_{[k:k+i+1]} = \mathbf{T}_{k+i}\mathbf{T}_{[k:k+i]}$. □

The following result about the sizes of entries in $\mathbf{T}_{[\ell:r]}$ will be useful later.

Lemma 3.6. *Let the recurrence in (10) be nice. Then for any $0 \le i, j \le t$, we have*

$$\|\mathbf{T}_{[\ell:r]}[i,j]\| \le \max((r-\ell-i)+j, 0).$$

Proof. Fix any arbitrary ℓ. We will prove the statement by induction on $r-\ell$. For the base case (i.e. when we are considering $\mathbf{T}_\ell$), the bounds follow from the definition of $\mathbf{T}_\ell$ and the our assumption on the sizes of $g_{\ell,j}(X)$ for $0 \le j \le t$. Assume the result is true for $r-\ell = \Delta \ge 0$.

Now consider the case of $r = \ell+\Delta+1$. In this case note that $\mathbf{T}_{[\ell:r]} = \mathbf{T}_{r-1} \cdot \mathbf{T}_{[\ell:r-1]}$. Now by the action of $\mathbf{T}_{r-1}$, the last t rows of $\mathbf{T}_{[\ell:r]}$ are the first t rows of $\mathbf{T}_{[\ell:r-1]}$ and the size claims for entries in those rows follows from the inductive hypothesis. Now note that for any $0 \le j \le t$, we have

$$\mathbf{T}_{[\ell:r]}[0,j] = \sum_{k=0}^{t} g_{r-1,k}(X) \cdot \mathbf{T}_{[\ell:r-1]}[k,j].$$

Since the recurrence is nice, we have that

$$\|\mathbf{T}_{[\ell:r]}[0,j]\| \le \max_{0\le k\le t} \|g_{r-1,k}\| + \|\mathbf{T}_{[\ell:r-1]}[k,j]\| \le \max_k (k+1) + (r-\ell-1-k) + j) = r-\ell+j,$$

as desired. □

The above along with induction implies that

Lemma 3.7. *Let the recurrence in* (1) *be nice. Then for every* $0 \le i < N$*, we have*

$$\|\mathbf{f}_i\| \le i,$$

where we think of $\mathbf{f}_i \in \mathbb{F}(X)$.

We note that our algorithms can handle the case when the initial $\|\mathbf{f}_j\|$ for $0 \le j \le t$ are larger than that specified in the above lemma.

4 Computing $\mathbf{A}^T\mathbf{b}$

We consider the problem of computing

$$\mathbf{c} = \sum_{i=0}^{N} \mathbf{b}[i]\mathbf{f}_i$$

for the standard recurrence defined by 3.2.

We will solve this problem by isolating the effect of the $\otimes$ operator and reducing the problem to a main computation just over $\mathbb{F}(X)$. Let $\mathbf{F} = \left[\mathbf{f}_0 \cdots \mathbf{f}_t\right]^T$. Lemma 3.5 implies that $\mathbf{f}_i$ is the last element of $\mathbf{T}_{[t:i+t]} \otimes \mathbf{F}$. Thus $\mathbf{c}$ is the last element of

$$\sum_{i=0}^{N-1} \mathbf{b}[i]\left(\mathbf{T}_{[t:i+t]} \otimes \mathbf{F}\right) \tag{11}$$

This can be rewritten as $\left(\sum_{i=0}^{N-1} \mathbf{b}[i]\mathbf{T}_{[t:i+t]}\right) \otimes \mathbf{F}$. Thus it suffices to compute the last row of this sum in parenthesis. It can be decomposed as

$$\sum_{i=0}^{N/2-1} \mathbf{b}[i]\mathbf{T}_{[t:i+t]} + \left(\sum_{N/2}^{N} \mathbf{b}[i]\mathbf{T}_{[N/2+t:i+t]}\right)\mathbf{T}_{[t:N/2+t]}$$

Note that both sums are the same problem but of size $N/2$, corresponding to both halves of the recurrence. This motivates defining $\mathbf{P}_{\ell,r}$ to be the last row of $\sum_{i=\ell}^{r-1} \mathbf{b}[i]\mathbf{T}_{[\ell+t:i+t]}$, so that the desired answer is simply $\mathbf{P}_{0,N} \otimes \mathbf{F}$. Furthermore, this quantity satisfies the relation

$$\mathbf{P}_{\ell,r} = \mathbf{P}_{\ell,m} + \mathbf{P}_{m,r}\mathbf{T}_{[\ell:m]} \tag{12}$$

for any $\ell \le m < r$, and thus can be computed with two recursive calls and a vector-matrix multiply over $\mathscr{R}(X)^{t+1}$.

Also, note that the $\mathbf{T}_{[\ell:r]}$ are independent of $\mathbf{b}$ and the relevant ones will be pre-computed.

Algorithm 1 TRANSPOSEMULT

```
Input: $\mathbf{b}, \mathbf{G}, a, k$
Input: $\mathbf{T}_{[bN/2^d:bN/2^d+N/2^{d+1}]}$ for $0 \le d < m$, $0 \le b < 2^d$
Output: $\mathbf{P}_{k,k+2^a}$ = First row of $\sum_{i=0}^{2^a-1} \mathbf{b}[k+i]\mathbf{T}_{[k:k+i]}$
1: If $2^a \le t$ then                                                    ▷ Base case
2:     Return $\begin{bmatrix} 0 & \cdots & \mathbf{b}[k+2^a-1] & \cdots & \mathbf{b}[k] \end{bmatrix}$
3: else
4:     Return TRANSPOSEMULT($\mathbf{b}, \mathbf{G}, a-1, k$) + TRANSPOSEMULT($\mathbf{b}, \mathbf{G}, a-1, k+2^{a-1}$)$\mathbf{T}_{[k:k+2^{a-1}]}$
```

Finally, observe that when $r - \ell \le t$, the last row of $\mathbf{T}_{[\ell:r]}$ is simply an indicator vector with a 1 in the $r-\ell$-th spot from the end (conceptually, the companion matrices act as a shift before reaching the recurrence width). Thus we stop the recurrence when the problem size gets below $t+1$.

We present the full details of the algorithm in Algorithm 1. The initial call is TRANSPOSEMULT($\mathbf{b}, \mathbf{G}, m, 0$) (recall that $N = 2^m$) and the relevant $\mathbf{T}_{[\ell:r]}$ are assumed to be precomputed, which we will describe next.

The above discussion implies that Algorithm 1 is correct:

Theorem 4.1. *Assuming all the required* $\mathbf{T}_{[\cdot,\cdot]}$ *are correctly computed,* TRANSPOSEMULT($\mathbf{b}, \mathbf{G}, m, 0$) *returns* $\mathbf{P}_{0,N}$.

4.1 Pre-processing time

In this section, we will see how we can efficiently compute the matrices of fractions $\mathbf{T}_{[bN/2^d:bN/2^d+N/2^{d+1}]}$ for $0 \le d < m$, $0 \le b < 2^d$. Since we have assumed that N is a power of 2, these ranges can be expressed in terms of dyadic strings; we need to pre-compute $\mathbf{T}_{[s]}$ for all strings $s \in \{0,1\}^*, |s| \le \lg N$ (where we interpret $[s]$ as the corresponding dyadic interval in $[0, N-1]$). All the required matrices can be computed in a natural bottom-up fashion:

Lemma 4.2. *We can pre-compute* $\mathbf{T}_{[s]}$ *for all strings* $s \in \{0,1\}^*, |s| \le \lg N$ *with* $O(N t^{\omega_{\mathbb{F}}} \log^2 N)$ *operations, where two* $n \times n$ *matrices over* $\mathbb{F}$ *can be multiplied with* $O(n^{\omega_{\mathbb{F}}})$ *many operations over* $\mathbb{F}$.

Proof. Fix an arbitrary s^* of length $\ell < \lg N$. We can compute $\mathbf{T}_{[s^*]} = \mathbf{T}_{[s^*0]} \cdot \mathbf{T}_{[s^*1]}$. Using the matrix multiplication algorithm, we have $O(t^{\omega_{\mathbb{F}}})$ fraction multiplications to compute, where the fractions are of size at most $\frac{N}{2^\ell} + t$ by Lemma 3.6. So computing $\mathbf{T}_{[s^*]}$ takes $O((N/2^\ell + t) t^{\omega_{\mathbb{F}}} \log N)$ operations. Computing $\mathbf{T}_{[s]}$ for all $|s| = \ell$ takes $O((N + t2^\ell) t^{\omega_{\mathbb{F}}} \log N)$, and computing all $\mathbf{T}_{[s]}$ takes $O(N t^{\omega_{\mathbb{F}}} \log^2 N)$, as desired.[5] □

The above implies that

Corollary 4.3. *Pre-processing for Algorithm 1 can be done with* $O(N t^{\omega_{\mathbb{F}}} \log^2 N)$ *many operations over* $\mathbb{F}$.

4.2 Runtime analysis

We now analyze the runtime of Algorithm 1 assuming that the pre-processing step (i.e. computing all the required $\mathbf{T}_{[\cdot:\cdot]}$) is already done. We do a simple recursive runtime analysis.

Lemma 4.4. *After pre-processing, Algorithm 1 needs* $O(t^2 N \log^2 N)$ *operations over* $\mathbb{F}$.

Proof. Let $T(N)$ denote the number of operations needed in the call TRANSPOSEMULT($\mathbf{b}, \mathbf{G}, m, 0$). Then it is easy to see that

$$T(N) = 2T(N/2) + O(\tau(N)),$$

where $\tau(N)$ is the number of operations taken in Step 4. We will show that $\tau(N) = O((t^2 N + t^3)\log N)$. Finally, note that $\tau(t+1) = O(t)$. This immediately implies the claim (note that the $O(t^3)$ term ultimately gets multiplied by N/t since we stop the recursion when the input size is $t+1$).

[5] In the last estimate we note that when $|s^*| \le \log_2(t+1)$, then all operations are over $\mathbb{F}$ and no fractions/polynomials are involved. Thus the $t^{\omega_R+1}2^\ell$ term only gets added up to $\ell = \log(N/t)$ and thus we do not pay an extra factor of t in the final analysis.

To see why $\tau(N) = O((t^2N + t^3)\log N)$, we first note that by Lemma 3.6, each element of vector $\mathbf{P}_{\ell,r}$ is a fraction of size at most $r - \ell$. Thus the vector-matrix multiplication in Step 4 performs $O(t^2)$ multiplications of fractions of size at most $N/2 + t$. Thus, the total number of operations used by the call to TRANSPOSEMULT($\mathbf{b}, \mathbf{G}, m, 0$) is $O(t^2(N + t)\log N)$, as desired. $\square$

4.3 Post-processing

We are not quite done yet with regard to computing $\mathbf{A}^T\mathbf{b}$ since after Algorithm 1, we still have to perform the step $\mathbf{P}_{0,N} \otimes \mathbf{f}$. Note that for the case of nice rational recurrence we have to do t multiplications over $\mathbb{F}(X)$ of size at most N: this can be done with $O(tN\log N)$ operations. Further for the case of an $\mathbf{R}$-matrix recurrence when $\mathbf{R}$ is (α, β)-Krylov efficient, implies that each of the t $\otimes$ operations can be done with $\tilde{O}(\beta N)$ operations (and $\tilde{O}(\alpha N)$ pre-processing operations).[6] These observations along with Corollary 4.3 and Lemma 4.4 imply the following result:

Theorem 4.5. *For any nice recurrence as in* (1)*, with $O(t^{\omega_{\mathbb{F}}} N \log^2 N)$ pre-processing operations, any $\mathbf{A}^T\mathbf{b}$ can be computed with $O(t^2 N \log^2 N)$ operations over $\mathbb{F}$. For an $\mathbf{R}$-matrix recurrence* (5) *that is nice and $\mathbf{R}$ is (α, β)-Krylov efficient, pre-processing and computation need additional $\tilde{O}(t\alpha N)$ and $\tilde{O}(t\beta N)$ operations respectively.*

We make a final remark on an optimization that can be done with "batch queries". First, note that equation 11 can be written as

$$\sum_{i=0}^{N-1} \begin{bmatrix} \mathbf{b}[i]0 & \cdots & 0 \end{bmatrix} \left(\mathbf{T}_{[t:i+t]} \otimes \mathbf{F}\right)$$

Now suppose we are given r vectors $\mathbf{b}_1, \ldots, \mathbf{b}_r$ and want to compute $\mathbf{A}\mathbf{b}_1, \ldots, \mathbf{A}\mathbf{b}_r$. Defining $\mathbf{B}_i = \begin{bmatrix} \mathbf{b}_1[i] & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{b}_r[i] & 0 & \cdots & 0 \end{bmatrix}$, it suffices to compute

$$\sum_{i=0}^{N-1} \mathbf{B}_i \mathbf{T}_{[t:i+t]} \tag{13}$$

We can call the above $\mathbf{P}_{0,N}$, defined the same way as before, and the recursive identity (12) still holds so we can still run Algorithm 1. The analysis in Lemma 4.4 showed that the bottleneck of the algorithm, which led to the t^2 runtime coefficient, is in the multiplication of a $1 \times t$ and $t \times t$ matrix. Thus for this batch query, the runtime will be $\tilde{O}(\alpha_r N)$ where α_r is the number of operations to multiply a $r \times t$ by $t \times t$ matrix.

Note in particular that $r = t$ multiplications can be computed simultaneously in $\tilde{O}(t^\omega N)$ time, and in general $\alpha_r \le \min(rt^2, (1 + r/t)t^\omega)$. For large numbers of queries, the amortized multiplication time can be considered to be $\tilde{O}(t^{\omega-1}N)$.

5 Overview of the rest of the paper

We present the rest of our main matrix-vector multiplication algorithms in Section 6. In particular, we present the algorithm to compute $\mathbf{Ab}$ in Section 6 and show that our algorithms have the same complexity guarantees for this problem as in Theorem 4.5. However, the arguments get a bit more tricky. First we show how to factor out a Krylov matrix in the case of matrix recurrences, to reduce Definition 3.4 to 3.3. Using a series of transformations we then reduce this to a basic polynomial recurrence 1.

Section 7 deals with the case when our recurrences have error. This is crucial for extending our results to work for low displacement rank matrices (and beyond). One solution is fairly simple: for a recurrence with the error matrix having rank r, we can reduce the problem with error to $r + 1$ instantiations of the problem with no errors and use our previous results as a black box. We also provide a more complicated approach with a reduced runtime. We also show in Theorem 11.4 that doing substantially better than this reduction would improve the state of the art algorithms for multipoint evaluation of multivariate polynomials.

[6]Indeed if $\mathbf{P}_{0,N}[j] = \sum_{i=0}^{N-1} p_i X^i$, then $\mathbf{P}_{0,N}[j](\mathbf{R}) \otimes \mathbf{f}_j = \mathscr{K}(\mathbf{R}, \mathbf{f}_j) \cdot \mathbf{p}$, where $\mathbf{p} = (p_0, \ldots, p_{N-1})$.

In Section 9, we show that our notion of recurrence width satisfies a hierarchy in a very strong sense: in particular to represent a matrix with recurrence width $t+1$ with a matrix of recurrence width t needs an error matrix of rank $\Omega(N)$. In Section 8 we present examples of some matrices that are Krylov efficient. We note that showing the latter for band matrices needs to use our algorithm from Section 6 for computing **Ab** in the rational case.

In Section 10, we consider some special cases of our general framework. In Sections 10.1 and 10.2, we show why low displacement rank matrices fall in our framework and how our results in some sense are simpler than known results for orthogonal polynomial transforms. In Section 10.3, we show how to speed up our algorithms for the case when the $g_{i,j}(X)$s only depend on j (i.e. the 'transition' matrix $\mathbf{T}_i$ is independent of i). In Section 10.4 we show how our general purpose algorithm can be used to compute Bernoulli numbers. In Section 10.5, we utilize our algorithm to compute the Jordan decomposition of some classes of matrices. In Section 10.6, we outline why our algorithms have good bit complexity and in particular, why they are competitive with special purpose algorithms [25] to compute Bernoulli numbers even in the bit complexity measure.

Finally in Section 11 we present the connections of our framework to multipoint evaluation of multivariate polynomials as well as coding theory. The aim of this section is two-fold. First this section presents matrices that have been studied in other contexts that have a small recurrence width in our setup (but do not easily fit into the existing notions of width). Second, this section shows that if the efficiency of our algorithms can be improved in some natural directions, then those improvements will immediately imply improvements over the state of the art algorithms for multipoint evaluation of multivariate polynomials over arbitrary fields (and with arbitrary evaluation points).

6 Computing Ab

In this section we will consider the problem of computing

$$\mathbf{c} = \mathbf{A}\mathbf{b}.$$

where the rows of **A** are defined by recurrence (10) as usual.

For now, we will make two simplifying assumptions:

1. Assume there is a vector $\mathbf{e} \in \mathbb{F}^N$ and elements $f_i(X) \in \mathcal{R}(X) : 0 \le i \le t$ such that $\mathbf{f}_i = f_i(X) \otimes \mathbf{e}$.
2. Assume that the $f_i(X)$ generated from $f_0(X), \dots, f_t(X)$ by (1) are polynomials.

The first assumption implies that $\mathbf{f}_i = f_i(X) \otimes \mathbf{e}$ for all $0 \le i < N$. The second assumption allows us to define the matrix $\mathbf{A}'$ consisting of coefficients of the $f_i(X)$. Furthermore, the problem of computing $(\mathbf{Ab})[i]$ is now

$$\begin{aligned}\langle \mathbf{f}_i, \mathbf{b} \rangle &= \left\langle f_i(X) \otimes \mathbf{e}, \mathbf{b} \right\rangle \\ &= \sum_{j=0}^{N-1} f_{i,j} \left\langle X^j \otimes \mathbf{e}, \mathbf{b} \right\rangle\end{aligned}$$

Thus, defining the vector $\mathbf{b}'$ where $\mathbf{b}'[i] = \left\langle X^j \otimes \mathbf{e}, \mathbf{b} \right\rangle$, the problem becomes computing $\mathbf{A}'\mathbf{b}'$. Hence using these assumptions, we can reduce the problem on a general recurrence (10) into one on a rational recurrence (that produces polynomials) (1), plus the computation of $\mathbf{b}'$.

For the remainder of this section, we first show how to solve the problem for a certain type of rational recurrence that satisfies assumptions 1 and 2. Next we address rational recurrences in a modulus, which allows us to remove assumption 2. Finally, we comment on removing assumption 1.

6.1 Basic rational recurrences

Consider the following recurrence:

$$D_{i+1}(X) f_{i+1}(X) = \sum_{j=0}^{t} n_{i,j}(X) \cdot f_{i-j}(X). \tag{14}$$

We assume that for every $0 \le i < N$ and $0 \le j \le t$, we have $\deg(D_i(X)) = \bar{d}$ and $\deg(n_{i,j}(X) \le d(j+1) + \bar{d}$ for non-negative integers $\bar{d}$ and d. We note that the above captures (4) (with potentially larger parameters).

Before we proceed, we make a useful definition. For every $0 \le \ell \le r < N$, define

$$D_{[\ell:r]}(X) = \prod_{k=\ell}^{r-1} D_k(X).$$

We now make an assumption on the recurrence in (14) to guarantee that assumption 2 holds. In particular, we will assume that for every $0 \le j \le t$, we have that

$$D_{[j+1:N]} \text{ divides } f_j(X) \text{ and } \deg(f_j) \le jd + (N-1)\bar{d}. \tag{15}$$

This assumption is sufficient to guarantee that the $f_i(X)$ are polynomials and allows us to consider the problem of computing $\mathbf{Ab}$ for a matrix $\mathbf{A}$ whose rows correspond to coefficients of f_i. Note that the prototypical example of multiplying by a matrix $\mathbf{A} \in \mathbb{F}^{N\times N}$ whose rows satisfy a polynomial recurrence falls exactly into this setting with parameters $\bar{d} = 0, d = 1$, and Section 8 uses this algorithm with $\bar{d} = 1, d = 0$.

The main idea is to use the following observation to compute $\mathbf{Ab}$. For any vector $\mathbf{u} \in \mathbb{F}^N$, define the polynomial

$$\mathbf{u}(X) = \sum_{i=0}^{N-1} u_i \cdot X^i.$$

Also define

$$\mathbf{u}^R = \mathbf{J} \cdot \mathbf{u},$$

for the *reverse* vector, where $\mathbf{J}$ is the 'reverse identity' matrix. With the above notations we have

Lemma 6.1. *For any vector $\mathbf{u}, \mathbf{v} \in \mathbb{F}^N$, we have $\langle \mathbf{u}, \mathbf{v} \rangle \left(\langle \mathbf{u}, \mathbf{v}^R \rangle\right)$ is the coefficient of degree $N-1$ in the polynomial $\mathbf{u}(X) \cdot \mathbf{v}^R(X)$ ($\mathbf{u}(X) \cdot \mathbf{v}(X)$ resp.).*

Proof. The proof follows from noting that the coefficient of X^{N-1} in $\mathbf{u}(X) \cdot \mathbf{v}^R(X)$ is given by

$$\sum_{j=0}^{N-1} \mathbf{u}[i] \cdot \mathbf{v}^R[N-1-i] = \sum_{i=0}^{N-1} \mathbf{u}[i] \cdot \mathbf{v}[i] = \langle \mathbf{u}, \mathbf{v} \rangle,$$

where we used that fact that $\left(\mathbf{v}^R\right)^R = \mathbf{v}$. □

To handle (14) with the assumption (15), we define a new recurrence motivated by the above observation. For every $0 \le i < N$, define

$$p_i(X) = \frac{f_i(X)\mathbf{b}^R(X)}{D_{[i+1:N]}(X)}. \tag{16}$$

Next, we argue that the polynomials $p_i(X)$ satisfy a recurrence with bounded recurrence width. In particular, we have that

Lemma 6.2. *For $i \ge t$ we have*

$$p_{i+1}(X) = \sum_{j=0}^{t} g_{i,j}(X) p_{i-j}(X),$$

where $g_{i,j(X)} \in \mathbb{F}[X]$ and $\deg(g_{i,j}(X)) \le (j+1)(\bar{d}+d)$.

Proof. Multiplying both sides of (14) by $\frac{\mathbf{b}^R(X)}{D_{[i+1:N]}(X)}$ along with (16) we get

$$\begin{aligned} p_{i+1}(X) &= \sum_{j=0}^{t} n_{i,j}(X) \cdot D_{[i-j+1:i+1]}(X) \cdot p_{i-j}(X) \\ &= \sum_{j=0}^{t} g_{i,j}(X) p_{i-j}(X), \end{aligned}$$

where we define $g_{i,j}(X) = n_{i,j}(X) \cdot D_{[i-j+1:i+1]}(X)$. The claim on the degree of $g_{i,j}(X)$ follows from the degree bounds on $n_{i,j}(X)$ and $D_i(X)$s. □

We note that assumption (15) implies that the starting functions $p_i(X) : 0 \le i \le t$ are polynomials (and not just rational functions) and Lemma 6.2 implies that all $p_i(X)$ are polynomials. A similar argument shows that the $f_i(X)$ are polynomials.

Lemma 6.3. *Under assumption* (15)*, the functions* $p_i(X)$ *defined by* (16) *are polynomials.*

We can use the degrees of the starting polynomials, together with the recurrence, to show degree bounds on all $f_i(X)$. It is straightforward to inductively show that $\deg f_i(X)/D_{[i+1:N]}(X) \le i(\bar{d}+d)$ using the fact that $\deg g_{i,j}(X) \le (j+1)(\bar{d}+d)$. Then

$$\deg(f_i(X)) \le i(\bar{d}+d) + \deg(D_{[i+1,N]}(X)) = i(\bar{d}+d) + (N-i-1)\bar{d} = (N-1)\bar{d} + id.$$

So the following lemma is true.

Lemma 6.4. *For every* $0 \le i < N$*, we have*

$$\deg(p_i(X)) \le i(\bar{d}+d) + \deg(\mathbf{b}^R(X)) \textit{ and } \deg(f_i(X)) \le (N-1)\bar{d} + id.$$

Thus $\mathbf{A}$ is a matrix of dimensions $N \times \overline{N}$ where

$$\overline{N} = (N-1)(\bar{d}+d) + 1.$$

Note that $\mathbf{A}$ is square when $\bar{d}+d = 1$; this includes the basic polynomial recurrence (such as the one orthogonal polynomials satisfy) as well as in applications such as Section 8. With these definitions, we will show how to compute the product $\mathbf{Ab}$ where $\mathbf{b}$ is a vector of length $\overline{N}$.

To obtain $\mathbf{Ab}$, by Lemma 6.1 and (16), we need to compute the coefficient of the monomial $X^{\overline{N}-1}$ for every $0 \le i < N$:

$$p_i(X) \cdot D_{[i+1:N]}(X).$$

By Lemma 3.5, equivalently, we need to compute the coefficient of the monomial $X^{\overline{N}-1}$ for every $0 \le i < N$:

$$\sum_{j=0}^{t} h_{i,j}^{(0)}(X) p_j(X) D_{[i+1:N]}(X). \tag{17}$$

Fix an $0 \le i < N/2$. Rewriting the above, $\mathbf{c}[i]$ is the coefficient of $X^{\overline{N}-1}$ in the polynomial

$$\sum_{j=0}^{t} \left(h_{i,j}^{(0)}(X) D_{[i+1:N/2]}(X)\right) \cdot \left(p_j(X) D_{[N/2:N]}(X)\right).$$

Now note that

$$\deg\left(h_{i,j}^{(0)}(X) D_{[i+1:N/2]}(X)\right) \le (i-j)(\bar{d}+d) + (N/2-i-1)\bar{d} \le \frac{\overline{N}}{2} - j(\bar{d}+d),$$

where the first inequality follows from the fact (that can be proven by induction) that $\deg(h_{i,j}^{(0)}(X)) \le (i-j)(\bar{d}+d)$ and the second inequality uses the fact that $i \le N/2-1$. This implies that to compute $\mathbf{c}[i]$ for any $0 \le i < N/2$, we only need the coefficients of X^k for $k \in \left[\frac{\overline{N}}{2} + j(\bar{d}+d), \overline{N}-1\right]$ in the polynomial

$$q_j(X) = p_j(X) D_{[N/2:N]}(X).$$

for $0 \le j \le t$.

In particular, we can first isolate the higher order $\overline{N}/2$ coefficients of the starting polynomials. For convenience, define the operator

$$\text{Extract}(f(X), n) = \frac{\left(f(X) \mod X^n\right) - \left(f(X) \mod X^{n/2}\right)}{X^{n/2}}$$

and for $0 \le j \le t$ let

$$p'_j(X) = \text{Extract}(q_j(X), \overline{N})$$

then note that $\mathbf{c}[i]$ is the coefficient of $\frac{\overline{N}}{2} - 1$ in the polynomial

$$\sum_{j=0}^{t} \left(h_{i,j}^{(0)}(X) D_{[i+1:N/2]}(X)\right) \cdot p'_j(X).$$

Comparing to equation (17), this is an problem of size $\overline{N}/2$ and we can recurse. We now consider the output element $\mathbf{c}[i + N/2]$ for $0 \le i < \overline{N}/2$. The argument is similar to the previous case. Again by Lemma 3.5), $\mathbf{c}[i + N/2]$ is the coefficient of the monomial $X^{\overline{N}-1}$ in the polynomial

$$\sum_{j=0}^{t} h_{i,j}^{(N/2)}(X) D_{[i+N/2+1:N]}(X) \cdot p_{N/2+j}(X).$$

By the same argument in the previous case, we have that

$$\deg\left(h_{i,j}^{(N/2)}(X) D_{[i+N/2+1:N]}(X)\right) \le \frac{\overline{N}}{2} - j(\bar{d} + d).$$

This implies that we need the coefficients of X^i for $i \in \left[\frac{\overline{N}}{2} + j(\bar{d}+d), \overline{N} - 1\right]$ in $p_{N/2+j}(X)$. If we define

$$p'_{N/2+j}(X) = \text{Extract}(p_{N/2+j}(X), \overline{N})$$

then $\mathbf{c}[i]$ is the coefficient of $\frac{\overline{N}}{2} - 1$ in the polynomial

$$\sum_{j=0}^{t} \left(h_{i,j}^{N/2)}(X) D_{[i+N/2+1:N]}(X)\right) \cdot p'_{N/2+j}(X).$$

Thus, we have reduced our problem to size $\overline{N}/2$ and we can recurse. Note that this assumes we have access to $p_{N/2+j}(X)$, which can be quickly computed assuming our usual pre-processing step.

The discussion above is formalized in Algorithm 2, the correctness of which follows from the above discussion. (The initial call is to $\text{MatrixVectMultRational}(p_0(X), \ldots, p_t(X), \overline{N}, m', 0)$, where we assume that $\overline{N} = 2^{m'}$.)

We argue that Algorithm runs efficiently. In particular,

Lemma 6.5. *A call to* $\text{MatrixVectMultRational}(p_0(X), \ldots, p_t(X), \overline{N}, m', 0)$ *takes* $O(t^2(\bar{d} + d)N \log^2 N)$ *many operations.*

Proof. If $T(\overline{N})$ is the number of operations needed for a call to Algorithm 2, then we will show that

$$T(\overline{N}) \le 2T(\overline{N}/2) + O(t^2 \overline{N} \log \overline{N})$$

with base case $T(t) \le O(t^3(\bar{d} + d)\log^2(t(\bar{d} + d)))$. This will prove the claimed runtime.

To see the base case note that computing each $q_i(X)$ involves $O(t)$ polynomial multiplications where the polynomials have degrees $O(t(\bar{d} + d))$, which implies $O(t^2 d \log^2(t(\bar{d} + d)))$ operations. Since there are $O(t)$ such $q_i(X)$ to compute, we have $T(t) \le O(t^3(\bar{d} + d)\log^2(t(\bar{d} + d)))$, as claimed.

Thus, to complete the argument we need to show that in a recursive call of size N we use $O(t^2(\bar{d} + d)\log N)$ many operations.

For the first recursive call, the runtime is dominated by the the number of operations taken to compute $q_j(X)$. Naively, this can be done with t multiplications of polynomials of degree $O(\overline{N})$, so computing all the $q_j(X)$ takes $O(t^2 \overline{N} \log^2 \overline{N})$ operations. For the second recursive call, the runtime is dominated by Step 9 is matrix vector multiplication with dimension $t + 1$ where each entry is a polynomial of degree $O(\overline{N})$. Hence, this steps also takes $O(t^2 \overline{N} \log \overline{N})$ many operations, as desired. $\square$

Thus, we have argued the following result:

Theorem 6.6. *For any nice recurrence as in* (14) *such that* $\max_i \deg(D_i(X)) \le \bar{d}$ *and* $\deg(n_{i,j}(X)) \le d(j+1) + \bar{d}$ *and satisfying* (15), *with* $O(t^{\omega_{\mathbb{F}}}(\bar{d} + d)N \log^2 N)$ *pre-processing operations, any* $\mathbf{Ab}$ *can be computed with* $O(t^2(\bar{d} + d)N \log^2 N)$ *operations over* $\mathbb{F}$.

Algorithm 2 MatrixVectMultRational

```
Input: $p_0(X),\dots,p_t(X),\Delta,a,k$ with the assumption (15)
Input: $\mathbf{T}_{\left[\frac{bN}{2^d}:\frac{bN}{2^d}+\frac{N}{2^{d+1}}\right]}$ for $0 \le d < m$ and $0 \le b < 2^d$
Input: $D_{[bN/2^d:bN/2^d+N/2^{d+1}]}(X)$ for $0 \le d < m$ and $0 \le b < 2^d$
Output: $\mathbf{c}$ such that $\mathbf{c}[i]$ is the coefficient of $X^{\Delta-1}$ of $\sum_{j=0}^{t} h_{i,j}^{(k)}(X)p_j(X)D_{[k+i+1:k+2^a]}(X)$ for $0 \le i < 2^a$
 1: $n \leftarrow 2^a$
 2: If $n \le t$ then                                                  ▷ Base case
 3:     For every $0 \le i < n$ do
 4:         $q_i(X) \leftarrow \sum_{j=0}^{t} h_{i,j}^{(k)}(X)p_j(X)D_{[k+i+1:k+n]}(X) = p_j(X)D_{[k+i+1:k+n]}$
 5:         $\mathbf{c}[i] \leftarrow$ coefficient of $X^{\Delta-1}$ in $q_i(X)$
 6: For every $0 \le j \le t$ do                       ▷ Do computation for the first recursive call
 7:     $p'_j(X) \leftarrow \text{Extract}(p_j(X)D_{[k+n/2:k+n]}(X), \overline{N})$
 8: $\mathbf{c}[k:k+n/2] \leftarrow$ MatrixVectMultRational$(p'_0(X),\dots,p'_t(X),\Delta/2,a-1,k)$
 9: $\left(p_{n/2+t}(X),\dots,p_{n/2}(X)\right)^T \leftarrow \mathbf{T}_{[k:k+n/2]}\left(p_t(X)\cdots,p_0(X)\right)^T$     ▷ Do computation for second recursive call
10: For every $0 \le j \le t$ do
11:     $p'_{n/2+j} \leftarrow \text{Extract}(p_{n/2+j}(X),\Delta)$
12: $\mathbf{c}[k+n/2:k+n] \leftarrow$ MatrixVectMultRational$(p'_{n/2}(X),\dots,p'_{n/2+t}(X),\Delta/2,a-1,k+n/2)$
13: Return $\mathbf{c}[k:k+n]$
```

6.2 Rational recurrences in a modulus

Suppose we have a recurrence (14) but the conditions (15) do not hold. In particular, it may not be the case that the $f_i(X)$ are polynomials, so our usual way of defining the coefficient matrix $\mathbf{A}$ does not make sense. However, we may still want to work with the functions $f_i(X)$. As an example, we may be interested in the projections $\sum_{j=0}^{N-1} b_j f_i(\alpha_j)$ - see section 10.2 for some connections. Note that if we are only interested in the evaluations of a functions at a point α, we can consider inverses in $\mathbb{F}[X]/(X-\alpha)$ instead of in $\mathbb{F}(X)$, i.e. $1/f(\alpha) = g(\alpha) \iff g = f^{-1} \pmod{X-\alpha}$. This motivates considering rational recurrences in a modulus:

$$D_{i+1}(X)f_{i+1}(X) \equiv \sum_{j=0}^{t} n_{i,j}(X)\cdot f_{i-j}(X) \pmod{M(X)}. \tag{18}$$

The above equation assumes that $M(X)$ is chosen so that all $D_i(X)$ are invertible in the mod, or $\gcd(D_i, M) = 1$ for all i. We formally define the above recurrence to mean that given $f_0(X),\dots,f_i(X)$, then $f_{i+1}(X)$ is defined to be the unique polynomial of minimal degree satisfying (18).

We also assume that the roots of $M(X)$ are known, which will be necessary in the matrix multiplication. In the context of $\mathbf{R}$-matrix recurrences, the primary motivation for considering this recurrence, this is equivalent to knowing the eigenvalues of $\mathbf{R}$.

The point of this recurrence is that it converts our applications involving rational functions into applications involving polynomials, whence we can apply the techniques of section 6.1 (after another step described in this section). Put another way, we will be able to remove the dependence on assumption 2. In general, the transformation from a rational function $f_i(X)$ to a vector $\mathbf{f}_i = f_i(X) \otimes \mathbf{e}$ is heavily dependent on the behavior of the $\otimes$ operator and must be considered on a case-by-case basis. Here are two core uses in line with our main interpretations of $\otimes$.

1. In the projection example, we suppose rational functions $f_i(X)$ satisfy recurrence (14) and want to calculate $\mathbf{Zb}$ where $\mathbf{Z}[i,j] = f_i(\alpha_j)$. If we instead define the $f_i(X)$ according to (18) with $M(X) = \prod(X-\alpha_j)$, then they have the same evaluations at the α_j. But now the f_i are polynomials, so we can factor $\mathbf{Z} = \mathbf{AV}^T$ where $\mathbf{A}$ is the coefficient matrix of the f_i and $\mathbf{V}$ is the Vandermonde matrix on the $\langle\alpha_j\rangle$.

2. Suppose we are in the setting of recurrence (5). We can instead consider the recurrence (18) with $M(X)$ set to the characteristic polynomial of $\mathbf{R}$. Since $M(\mathbf{R}) = 0$, the modular recurrence will produce a set of polynomials

$f_i(X)$ such that the resulting $\mathbf{f}_i = f_i(X) \otimes \mathbf{e}$ will be the same. This in fact generalizes the preceding application by taking $\mathbf{R} = \mathrm{diag}\langle \alpha_j \rangle$ and $\mathbf{e} = \mathbf{1}$ (the all 1s vector).

We now show, given (18), how to compute $\mathbf{Ab}$ and $\mathbf{A}^T\mathbf{b}$ where $\mathbf{A}$ is the coefficient matrix of the $f_i(X)$. Assume that $M(X)$ has degree N.

Define $D(X) = D_{[0:N]}(X)$ and $C(X)$ to be its inverse modulo $M(X)$. Note that $D(X)$ can be computed in time $O(dN\log^2 dN)$ by divide-and-conquer, and $C(X)$ can also be computed in $\widetilde{O}(N)$ operations using the fast Euclidean Algorithm.

Define $g_i(X)$ satisfying the same recurrence as the $f_i(X)$, but with starting conditions scaled by $D(X)$, and over $\mathbb{F}[X]$ instead of $\mathbb{F}[X]/(M(X))$, i.e.

$$\begin{aligned} g_i(X) &= D(X) f_i(X) && 0 \le i \le t \\ D_{i+1}(X) g_{i+1}(X) &= \sum_{j=0}^{t} n_{i,j}(X) \cdot g_{i-j}(X) && i > t \end{aligned}$$

Note that this recurrence satisfies assumptions (15) so we can use the results of section 6.1. Also, these polynomials satisfy a close relation with the $f_i(X)$.

Lemma 6.7. *For all i, $g_i(X)C(X) \equiv f_i(X) \pmod{M(X)}$.*

Proof. We can show this inductively. For $0 \le i \le t$, by construction $M(X)$ divides $g_i(X)C(X) - f_i(X) = f_i(X)(C(X)D(X) - 1)$. For $i > t$, multiply the recurrence for $g_i(X)$ by $C(X)$ and subtract (18). We are left with

$$D_{i+1}(X)\left(g_{i+1}(X)C(X) - f_{i+1}(X)\right) \equiv \sum_{j=0}^{t} n_{i,j}(X) \cdot \left(g_{i-j}(X)C(X) - f_{i-j}(X)\right) \pmod{M(X)}.$$

Inductively, $M(X)$ divides the RHS, so it divides the LHS as well. But it is relatively prime to $D_{i+1}(X)$, proving the claim. □

Finally, we show how to compute the queries $\mathbf{A}^T\mathbf{b}$ and $\mathbf{Ab}$.

The answer to $\mathbf{A}^T\mathbf{b}$ is just the coefficients of the polynomial $\sum b_i f_i(X)$. This sum is equal to $\sum b_i g_i(X)C(X) \pmod{M(X)}$. Let $\mathbf{A}'$ be the coefficient matrix for the $g_i(X)$ - note that it has dimensions $N \times K$ where $K = \max_i \deg(g_i)$ is clearly bounded by $(d + \bar{d})N$. Then it suffices to compute $\mathbf{A}'^T\mathbf{b}$, then convolve with the coefficients of $C(X)$ and reduce $\pmod{M(X)}$.

To compute $\mathbf{Ab}$, we define $\mathbf{Z}$ to be the $N \times N$ evaluation matrix at the roots of $M(X)$, i.e. $\mathbf{Z}[i,j] = f_i(\alpha_j) = g_i(\alpha_j)C(\alpha_j)$. Define $\mathbf{D} = \mathrm{diag}\langle C(\alpha_j)\rangle$ and $\mathbf{V}, \mathbf{V}'$ to be the Vandermonde matrices over roots $\alpha_1, \dots, \alpha_N$ of dimensions $N \times N$ and $N \times K$ respectively. Then the definition of $\mathbf{Z}$ is equivalent to the identities $\mathbf{A} = \mathbf{AV}^T$ and $\mathbf{A} = \mathbf{A}'\mathbf{V}'^T\mathbf{D}$. Therefore multiplying by $\mathbf{A} = \mathbf{A}'\mathbf{V}'^T\mathbf{D}\mathbf{V}^{-T}$ is bottlenecked by and thus has the same asymptotic runtime as multiplying by $\mathbf{A}'$.

So both types of multiplications reduce to the corresponding multiplications for rational recurrences without a modulus, which are handled by Algorithm 2.

6.3 Removing the assumptions

Our description of the core $\mathbf{Ab}$ algorithm, described in Section 6.1, made the simplifying assumptions that $\mathbf{f}_i = f_i(X) \otimes \mathbf{e}$ for $0 \le i \le t$, and that the $f_i(X)$ are polynomials. These assumptions do hold often - for example, trivially when $\otimes$ is interpreted as rational multiplication - but not always. However, even when they do not hold, we can perform simple transformations to ensure that the algorithm still works.

First, when assumption 2 does not hold, we use the technique of Section 6.2 to reinterpret the recurrence in a modulus so that the $f_i(X)$ are polynomials, before applying the main algorithm.

So assumption 2 is true; now suppose assumption 1 does not hold. Note that

$$\mathbf{f}_i = \dots + 0 \otimes \mathbf{f}_{i-1} + 1 \otimes \mathbf{f}_i + 0 \otimes \mathbf{f}_{i+1} + \dots$$

so we can define the $\mathbf{f}$ as the sum of t recurrences, where the jth recurrence has initial conditions $\delta_{i,j} \otimes \mathbf{f}_i : 0 \le i \le t$. Formally,

$$\begin{aligned}
\mathbf{c}[i] &= \langle \mathbf{f}_i, \mathbf{b} \rangle \\
&= \sum_{j=0}^{t} \left\langle h_{i,j}^{(0)}(X) \otimes \mathbf{f}_j, \mathbf{b} \right\rangle && (19) \\
&= \sum_{j=0}^{t} \sum_{\ell=0}^{N-1} h_{i,j,\ell}^{(0)} \left\langle X^{\ell} \otimes \mathbf{f}_j, \mathbf{b} \right\rangle && (20) \\
&= \sum_{j=0}^{t} \sum_{\ell=0}^{N-1} h_{i,j,\ell}^{(0)} \mathbf{b}_j[\ell]. && (21)
\end{aligned}$$

In the above (19) follows from Lemma 3.5, (20) follows by defining

$$h_{i,j}^{(k)}(X) = \sum_{\ell=0}^{N-1} h_{i,j,\ell}^{(k)} \cdot X^{\ell}$$

and (21) follows by defining the vectors $\mathbf{b}_j$ such that

$$\mathbf{b}_j[\ell] = \left\langle X^{\ell} \otimes \mathbf{f}_j, \mathbf{b} \right\rangle. \tag{22}$$

(21) implies that to compute $\mathbf{Ab}$ it is enough to be able to compute

$$\sum_{j=0}^{t} \mathscr{H}_j^{(0)} \mathbf{b}_j, \tag{23}$$

where $\mathscr{H}_j^{(0)}[i,\ell] = h_{i,j,\ell}^{(0)}$ and we can pre-compute the $\mathbf{b}_j$.

Note that by Lemma 3.5, each $\mathscr{H}_j^{(0)}$ satisfies (1) and (2) so we can run $t+1$ passes of Algorithm 2.

However, we can get the same asymptotic result with a single pass but modifying the initial polynomials to be a sum. Applying Lemma 6.1 to find $\mathbf{c}[i]$ means the only change necessary is replacing the numerator of definition (16) with $\sum_{j=0}^{t} h_{i,j}^{(0)}(X)\mathbf{b}_j^R(X)$. Thus the only change to Algorithm 2 is the computation of initial polynomials $p_0(X), \ldots, p_t(X)$ which now takes time $\widetilde{O}(t^2(\bar{d}+d)N)$ instead of $\widetilde{O}(t(\bar{d}+d)N)$, and the algorithm is asymptotically the same.

Finally, to run Algorithm 2, we need to compute the vectors $\mathbf{b}_0, \ldots, \mathbf{b}_t$ according to (22). We consider two cases corresponding to (1) and (5). For the case of nice rational recurrences, we note that $X^k \cdot f_j(X)$ just amounts to shifting the coefficient vector $\mathbf{f}_j$ by k. In particular, one can compute $\mathbf{b}_j$ from $\mathbf{b}$ by multiplying a triangular Toeplitz matrix (that depends on $\mathbf{f}_j$) with $\mathbf{b}$, which can be done with $O(N \log N)$ operations and hence, all the $\mathbf{b}_j$'s can be computed with $O(Nt \log N)$ operations. Further for the case of an $\mathbf{R}$-matrix recurrence when $\mathbf{R}$ is (α, β)-Krylov efficient, implies that each of the t $\otimes$ operations can be done with $\tilde{O}(\beta N)$ operations (with $\tilde{O}(\alpha N)$ pre-processing operations for each).[7] Together with the fact that the pre-processing needed for Algorithm 2 is exactly the same as in Section 4.1, implies the full result.

Theorem 6.8. *For any nice recurrence as in* (1)*, with* $O(t^{\omega_{\mathbb{F}}} N \log^2 N)$ *pre-processing operations, any* $\mathbf{Ab}$ *can be computed with* $O(t^2 N \log^2 N)$ *operations over* $\mathbb{F}$*. For an* $\mathbf{R}$*-matrix recurrence* (5) *that is nice and* $\mathbf{R}$ *is* (α,β)*-Krylov efficient, pre-processing and computation need additional* $\tilde{O}(t\alpha N)$ *and* $\tilde{O}(t\beta N)$ *operations respectively.*

7 Recurrences with Error

We now modify our approach to handle a low-rank error term in our recurrence. In particular, consider the following recurrence relations:

$$\mathbf{f}_{i+1} = \sum_{j=0}^{t} g_{i,j}(X) \otimes \mathbf{f}_{i-j} + \sum_{\ell=1}^{r} c_{i,\ell}(X) \otimes \mathbf{d}_{\ell} \tag{24}$$

[7] Note that $\mathbf{b}_j = \mathscr{K}(\mathbf{R}, \mathbf{f}_j) \cdot \mathbf{b}$.

for $\mathbf{d}_\ell \in \mathbb{F}^N$ and $c_{i,\ell}$ of size at most d, where $g_{i,j}(X)$ has size at most $d(j+1)$. These additional vectors represent a rank-r error in our recurrence. Such a recurrence is said to have *recurrence error width* of (t,r).

We present two approaches for solving a recurrence of this type. The first approach is a direct reduction to our standard recurrence (10), where we fold the error terms into the recurrence width. We can insert 'dummy' polynomials into the sequence to capture the errors. In particular, we insert r rows consisting of the $\mathbf{d}_\ell$ between every t of the original $\mathbf{f}_i$. Now note that every row of this resulting recurrence can be expressed in terms of the previous $t+r$ rows. Thus this results in a sequence of length $O(N\frac{t+r}{t})$ that follows a recurrence of width $t+r$. We can then apply our previous algorithms to compute $\mathbf{A}^T\mathbf{b}$ and $\mathbf{A}\mathbf{b}$. Note that the problem size depends on the width N of the matrix and not the height $N(1+r/t)$, because we avoid any computations involving rows corresponding to the dummy polynomials since we already know them. Therefore the algorithms take $O((r+t)^2 N\log^2 N)$ operations and $O((r+t)^{\omega_\mathbb{F}} N\log^2 N)$ preprocessing. We also remark that the r error terms can be separated and each handled with a recurrence of width $t+1$, leading to runtime and preprocessing $O(rt^2N\log^2 N)$ and $O(t^{\omega_\mathbb{F}}N\log^2 N)$ instead.

This results in a simple and direct reduction to our previous algorithms to handle recurrences with error, and also showcases that our basic notion of recurrences (10) is powerful enough to capture more a more complicated recurrence (24). However, this reduction is somewhat loose (i.e. a recurrence of width $t+r$ is strictly more powerful than a recurrence of width (t,r)) and we can do better.

Next, we show how to handle the error terms more precisely to achieve a better algorithm runtime. This approach will also utilize on the previous algorithms, but as a subroutine of a larger divide-and-conquer. First we will modify the work-horse lemma to handle all the error terms.

Consider a recurrence defined by (24) again. We will switch to using a presentation where we consider dummy starting conditions $\mathbf{f}_{-t-1} = \cdots = \mathbf{f}_{-1} = 0$, and $\mathbf{f}_0, \ldots, \mathbf{f}_t$ are just elements defined by the recurrence with error: e.g. $\mathbf{f}_t = \sum_{j=0}^{t} 0 \otimes \mathbf{f}_{t-1-j} + \mathbf{f}_t$ (compare to (24)).

In the style of the work-horse lemma, the recurrence (24) can be written as

$$\begin{bmatrix} \mathbf{f}_{i+1} \\ \vdots \\ \mathbf{f}_{i-t+1} \end{bmatrix} = \mathbf{T}_i \otimes \begin{bmatrix} \mathbf{f}_i \\ \vdots \\ \mathbf{f}_{i-t} \end{bmatrix} + \begin{bmatrix} c_{i,1} & \cdots & c_{i,r} \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{bmatrix} \otimes \begin{bmatrix} \mathbf{d}_1 \\ \vdots \\ \mathbf{d}_r \end{bmatrix} \tag{25}$$

for $0 \le i < N$. Let that $c_{i,\ell}$ matrix be denoted $\mathbf{C}_i$. Composing (25) and using $\mathbf{f}_{-t-1} = \cdots = \mathbf{f}_{-1} = \mathbf{0}$, we get an analog to the work-horse lemma

$$\begin{bmatrix} \mathbf{f}_i \\ \vdots \\ \mathbf{f}_{i-t} \end{bmatrix} = \left(\mathbf{T}_{[0:i]}\mathbf{C}_0 + \mathbf{T}_{[1:i]}\mathbf{C}_1 + \cdots + \mathbf{T}_{[i:i]}\mathbf{C}_i\right) \otimes \begin{bmatrix} \mathbf{d}_1 \\ \vdots \\ \mathbf{d}_r \end{bmatrix}$$

We can use this work-horse lemma to compute $\mathbf{A}\mathbf{b}$ and $\mathbf{A}^T\mathbf{b}$ with a divide-and-conquer algorithm. We will show $\mathbf{A}^T\mathbf{b}$ here.

Let $\mathbf{B}_i = \begin{bmatrix} \mathbf{b}[i] & 0 & \cdots & 0 \end{bmatrix} \in \mathbb{F}^{t+1}$, so that $\mathbf{b}[i]\mathbf{f}_i = \mathbf{B}_i \begin{bmatrix} \mathbf{f}_i & \cdots & \mathbf{f}_{i-t} \end{bmatrix}^T$. Therefore, the query $\mathbf{A}\mathbf{b}$ or $\sum \mathbf{b}[i]\mathbf{f}_i$ is equal to

$$\sum_{i=0}^{N-1} \mathbf{B}_i \sum_{j=0}^{i} \mathbf{T}_{[j:i]}\mathbf{C}_j \otimes \mathbf{D} = \left(\sum_{0\le j\le i<N} \mathbf{B}_i \mathbf{T}_{[j:i]} \mathbf{C}_j \right) \otimes \mathbf{D}$$

where $\mathbf{D} = \begin{bmatrix} \mathbf{d}_1 & \cdots & \mathbf{d}_r \end{bmatrix}^T$.

We will drop the $\otimes\mathbf{D}$ and perform it at the end, so we only care about the sum, which can be computed using divide and conquer. We can break the sum into

$$\begin{aligned} \sum_{0\le j\le i<N} \mathbf{B}_i\mathbf{T}_{[j:i]}\mathbf{C}_j &= \sum_{0\le j\le i<N/2} \mathbf{B}_i\mathbf{T}_{[j:i]}\mathbf{C}_j + \sum_{j<N/2, i\ge N/2} \mathbf{B}_i\mathbf{T}_{[j:i]}\mathbf{C}_j + \sum_{N/2\le j\le i<N} \mathbf{B}_i\mathbf{T}_{[j:i]}\mathbf{C}_j \\ &= \sum_{0\le j\le i<N/2} \mathbf{B}_i\mathbf{T}_{[j:i]}\mathbf{C}_j + \left(\sum_{i\ge N/2} \mathbf{B}_i\mathbf{T}_{[N/2:i]}\right)\left(\sum_{j<N/2} \mathbf{T}_{[j:N/2]}\mathbf{C}_j\right) + \sum_{N/2\le j\le i<N} \mathbf{B}_i\mathbf{T}_{[j:i]}\mathbf{C}_j \end{aligned}$$

Note that the first sum corresponds to $\mathbf{f}_0, \ldots, \mathbf{f}_{N/2-1}$ and the other two correspond to $\mathbf{f}_{N/2}, \ldots, \mathbf{f}_{N-1}$. Also note that the first and last terms are self-similar of half the size. Thus they can be computed with recursive calls, and we only have to worry about the middle product.

We just need to compute the left and right sides of this product. Note that the left product is exactly equation (13) which is the main problem addressed in Section 4. It can be computed in $O(t^2 N \log^2 N)$ time, after seeing the query $\mathbf{b}$. The right product is again equivalent to equation (13), up to transpose and different indices on the transition matrix ranges. Since the $\mathbf{C}_i$ are known upfront, these sums can be precomputed in $O(\alpha_r N \log^2 N)$ operations, where α_r is the time it takes to multiply a $r \times t$ by $t \times t$ matrix.

Given the left and right products, their product is found by a $1 \times t$ by $t \times r$ matrix multiplication over polynomials of degree $O(N)$, which takes $O(rtN \log N)$ operations. Performing the entire divide-and-conquer algorithm incurs an additional $\log N$ factor. Thus the total runtime is $O((t^2 + rt)N \log^2 N)$.

The product $\mathbf{Ab}$ can be computed with similar modifications, which we will provide an overview of. First we make a high level observation about the $\mathbf{A}^T\mathbf{b}$ algorithm. We broke the problem into three pieces, two of which were identical problems of half the size, so that it sufficed to compute the final one. We can do the same with $\mathbf{Ab}$. Formally, we can define $\mathbf{f}'_i$ to be the first row of $\left(\sum_{j=0}^{N/2-1} \mathbf{T}[j : i + N/2]\mathbf{C}_j\right) \otimes \mathbf{D}$, which is the part of $\mathbf{f}_{i+N/2}$ not captured by the recursive subproblem. To compute $\mathbf{A}^T\mathbf{b}$, we needed $\sum_{i=0}^{N/2-1} \mathbf{b}[i]\mathbf{f}'_i = \left(\sum_{j<N/2, i\ge N/2} \mathbf{B}_i \mathbf{T}_{[j:i]}\mathbf{C}_j\right) \otimes \mathbf{D}$, and the sum decomposed into a product where one side was exactly the problem solved in Section 4. To compute $\mathbf{Ab}$, we need $\langle \mathbf{f}'_i, \mathbf{b}\rangle = \langle \left(\mathbf{T}_{[N/2:N/2+i]} \sum_{j=0}^{N/2-1} \mathbf{T}[j : N/2]\mathbf{C}_j\right) \otimes \mathbf{D}, \mathbf{b}\rangle$ for $0 \le i < N/2$. Note that by identity (7), we can fold the sum into $\mathbf{D}$ so that we need to compute $\langle \mathbf{T}_{[N/2:N/2+i]} \otimes \mathbf{D}', \mathbf{b}\rangle$. This is precisely the problem solved in Section 6.

Therefore we have shown

Theorem 7.1. *For any nice recurrence with error, with $O(t^{\alpha_r} N \log^2 N)$ pre-processing operations, any $\mathbf{Ab}$ or $\mathbf{A}^T\mathbf{b}$ can be computed with $O(t(r + t)N \log^2 N)$ operations over $\mathbb{F}$. For an $\mathbf{R}$-matrix recurrence with error that is nice and $\mathbf{R}$ is (α, β)-Krylov efficient, pre-processing and computation need additional $\tilde{O}(t\alpha N)$ and $\tilde{O}(t\beta N)$ operations respectively.*

Note that α_r, the time it takes to multiply a $r \times t$ by $t \times t$ matrix, is bounded by $O((1 + k/t)t^{\omega})$ operations. Thus the pre-processing computation can be bounded by $O(rt^{\omega-1} + t^{\omega})$ operations.

We remark that this algorithm recovers the bounds in Theorems 6.8 and 4.5 when $r = O(t)$. In particular, we lose nothing by formulating a recurrence (1) as a recurrence with $t+1$ errors: each starting condition $\mathbf{f}_j : 0 \le j \le t$ is an error polynomial with corresponding coefficients $c_{i,j} = \delta_{i,j}$.

8 Krylov Efficiency

Recall that given a matrix $\mathbf{M} \in \mathbb{F}^{N\times N}$ and a vector $\mathbf{y} \in \mathbb{F}^N$, the *Krylov matrix of* $\mathbf{M}$ *generated by* $\mathbf{y}$ (denoted by $\mathcal{K}(\mathbf{M}, \mathbf{y})$) is the $N \times N$ matrix whose ith column for $0 \le i < N$ is $\mathbf{M}^i \cdot \mathbf{y}$. We say that $\mathbf{M}$ is (α, β)-*Krylov efficient* if for every $\mathbf{y} \in \mathbb{F}^N$, we have that $\mathbf{K} = \mathcal{K}(\mathbf{M}, \mathbf{y})$ admits the operations $\mathbf{Kx}$ and $\mathbf{K}^T\mathbf{x}$ (for any $\mathbf{x} \in \mathbb{F}^N$) with $\tilde{O}(\beta N)$ many operations (with $\tilde{O}(\alpha N)$ pre-processing operations).

First, we show how Krylov efficiency can be reduced to Jordan efficiency. In section 10.5, we define a matrix $\mathbf{M}$ to have an efficient Jordan decomposition if it has a decomposition $\mathbf{M} = \mathbf{AJA}^{-1}$ such that $\mathbf{A}$ and $\mathbf{A}^{-1}$ admit superfast matrix-vector multiplication. Suppose that $\mathbf{R}$ is Jordan efficient. Observe that the Krylov multiplication can be expressed as

$$\begin{aligned}
\mathcal{K}(\mathbf{R}, \mathbf{y})\mathbf{x} &= \sum_{i=0}^{N-1} \mathbf{x}[i](\mathbf{R}^i\mathbf{y}) \\
&= \left(\sum_{i=0}^{N-1} \mathbf{x}[i]\mathbf{R}^i\right)\mathbf{y} \\
&= \mathbf{x}(\mathbf{R})\mathbf{y}
\end{aligned}$$

where we define the function $\mathbf{x}(X) = \sum \mathbf{x}[i]X^i$.

Note that $\mathbf{x}(X)$ is analytic and the multi-point Hermite-type evaluation problem on it is computable in $\tilde{O}(N)$ time [38] [39]. Thus the Krylov multiplication can be performed using Lemma 10.11, and Krylov efficiency of $\mathbf{R}$ reduces exactly to Jordan efficiency with the same complexity bounds. Therefore all Jordan-efficient matrices are also Krylov-efficient.

However, this reduction is clearly one way– Jordan efficiency is stronger than Krylov efficiency, but the latter problem has more structure that we can take advantage of. In this section, we will show that the class of banded triangular matrices are Krylov efficient by showing that the Krylov matrix itself satisfies a recurrence to which we can apply our techniques.

We remark that the application to recurrence width only requires $\mathbf{Kx}$ to be fast, but Krylov efficiency is a quite fundamental concept that is useful in other contexts. The product $\mathbf{Kx} = \sum \mathbf{x}[i]\mathbf{M}^i\mathbf{y}$ is naturally related to applications involving Krylov subspaces, matrix polynomials, and so on; for us, Krylov efficiency arises from needing to evaluate polynomials at a matrix. The product $\mathbf{K}^T\mathbf{b} = [\mathbf{b}\cdot\mathbf{x}, \mathbf{b}\cdot\mathbf{Ax}, \mathbf{b}\cdot\mathbf{A}^2\mathbf{x}, \ldots]$ is also useful; for example, it is the first step in the Wiedemann algorithm for computing the minimal polynomial or kernel vectors of a matrix $\mathbf{A}$ [30].

8.1 Krylov Efficiency of triangular banded matrices

Let $\mathbf{M}$ be an upper triangular Δ-banded matrix, i.e. all values other than $\mathbf{M}[i,\ell]$ for $i \le \ell < i+\Delta$ are zero. Let $\mathbf{y}$ be an arbitrary vector and let $\mathbf{K}$ denote the Krylov matrix of $\mathbf{M}$ with respect to $\mathbf{y}$.

We will show that $\mathbf{K}$ satisfies a rational recurrence with recurrence error width of $(\Delta, 1)$. Note that these results also hold for lower triangular matrices.

Define polynomials

$$f_i(X) = \sum_{j=0}^{N-1} \mathbf{K}[i,j]\cdot X^j$$

Let $\mathbf{F} = \begin{bmatrix} f_0(X) \\ \vdots \\ f_{N-1}(X) \end{bmatrix}$. We can alternatively express this as

$$\mathbf{F} = \sum_{j=0}^{N-1} \mathbf{K}[:,j]X^j = \sum_{j=0}^{N-1} (\mathbf{M}^i\mathbf{y})X^i = \left(\sum_{j=0}^{N-1} (\mathbf{M}X)^i\right)\mathbf{y}$$

Multiplying by $\mathbf{I} - \mathbf{M}X$, we get the equation

$$(\mathbf{I} - \mathbf{M}X)\mathbf{F} = \mathbf{y} - (\mathbf{M}X)^N\mathbf{y} \tag{26}$$

Therefore it is true that

$$(\mathbf{I} - \mathbf{M}X)\mathbf{F} \equiv \mathbf{y} \pmod{X^N} \tag{27}$$

and furthermore, $\mathbf{F}$ can be defined as the unique solution of equation (27) because $\mathbf{I} - \mathbf{M}X$ is invertible in $\mathbb{F}[X]/(X^N)$ (since it is triangular and its diagonal is comprised of invertible elements $(1 - \mathbf{M}[i,i]X)$).

But equation (27) can be interpreted as a rational recurrence with error. Explicitly expanding (27), the $f_i(X)$ satisfy

$$(1 - \mathbf{M}[i+1,i+1]X)f_{i+1}(X) \equiv \sum_{j=0}^{\Delta} \mathbf{M}[i+1,i-j]Xf_{i-j}(X) + \mathbf{y}[i] \pmod{X^N}$$

Therefore the multiplications $\mathbf{Kx}$ and $\mathbf{K}^T\mathbf{x}$ can be computed using the techniques of Section 6 and Section 7.

Theorem 7.1, then implies that

Theorem 8.1. *Any triangular Δ-banded matrix is $(\Delta^\omega, \Delta^2)$-Krylov Efficient.*

9 Hierarchy of Recurrence

In this section we show that a $t+1$-term recurrence cannot be recovered by a t-term recurrence, showing a clear hierarchy among the matrices that satisfy our recurrence. Fix an arbitrary N. We will be looking at the simplest form of our recurrence: a polynomial family $f_0, \ldots, f_{N-1}$ such that

$$f_{i+1}(X) = \sum_{j=0}^{t} g_{i,j}(X)f_{i-j}(X)$$

where $\deg(g_{i,j}) \le j$. Define $P(t)$ to be all families of N polynomials that satisfy our recurrence of size t. For simplicity, we assume that all polynomials have integer coefficients. For any polynomial family $f \in P(t)$, we define the matrix $\mathbf{M}(f)$ that contains the coefficients of the polynomials as its elements, as in Section 6. To show the hierarchy of matrices, we will show that no family in $P(t)$ can approximate the mapping specified by a particular family in $P(t+1)$.

Theorem 9.1. *For every $t \ge 1$, there exists an $f \in P(t+1)$ such that for every $g \in P(t)$*

$$\left\|\mathbf{M}(f)\cdot\mathbf{1} - \mathbf{M}(g)\cdot\mathbf{1}\right\|_2^2 = \Omega\left(\frac{N}{t}\right).$$

Proof. We are going to choose f such that $f_i(X) = X^i$ if $i = k(t+1)$ for some $k \ge 0$ and $f_i(X) = 0$ otherwise. Note that $f \in P(t+1)$. Let $\mathbf{c} = \mathbf{M}(f)\cdot\mathbf{1}$. In particular,

$$\mathbf{c}[i] = \begin{cases} 1 & \text{if } i = k(t+1) \text{ for some } k \ge 0 \\ 0 & \text{otherwise} \end{cases}.$$

Fix an arbitrary $g \in P(t)$. Let $\mathbf{c}' = \mathbf{M}(g)\cdot\mathbf{1}$; note that $\mathbf{c}'[i] = g_i(1)$. Note that if any t consecutive $g_i(1),\dots,g_{i+t-1}(1)$ are 0, the t-term recurrence implies that all subsequent polynomials in family uniformly evaluate to 0 at 1. So we have two cases: (1) $g_i(1) = 0$ for all $i > N/2$ or (2) for each k, there exists an i such that $k(t+1) < i < (k+1)(t+1) \le N/2$ and $g_i(1) \ne 0$. In the first case $\mathbf{c}'[i] = 0$ for all $i > N/2$, and $\|\mathbf{c}-\mathbf{c}'\|_2^2 \ge \frac{N}{2(t+1)}$. Similarly, in the second case $\mathbf{c}'[i] \ne 0$ and $\mathbf{c}[i] = 0$ for each of the specified i, implying once again that $\|\mathbf{c}-\mathbf{c}'\|_2^2 \ge \frac{N}{2(t+1)}$. □

Corollary 9.2. *For every $t \ge 1$, there exists an $f \in P(t+1)$ such that for every $g \in P(t)$, $rank(\mathbf{M}(f) - \mathbf{M}(g)) = \Omega(\frac{N}{t})$.*

Proof. Let $\mathbf{H} = \mathbf{M}(f) - \mathbf{M}(g)$. Once again, we define f such that $f_i(X) = X^i$ if $i = k(t+1)$ for some $k \ge 0$ and $f_i(X) = 0$ otherwise. Once again, we have two cases. First, $deg(g_i(X)) < i$ for all $i = k(t+1), i > N/2$. Then $\mathbf{H}[i,i] = 1$ for all $i = k(t+1), i > N/2$, implying $rank(\mathbf{H}) \ge \frac{N}{2t}$. In the second case, we rely on the degree bound of the transition polynomials; in particular, if $g_{i+1}(X) = \sum_{j=0}^{t} h_{i,j}(X) g_{i-j}(X)$, we bound $deg(h_{i,j}(X)) \le j$. If $deg(g_i(X)) = i$ for some $i = k(t+1), i > N/2$, we know that for each value of k such that $(k+1)(t+1) < N/2$, there exists a j such that $k(t+1) < j < (k+1)(t+1)$ and $deg(g_j(X)) = j$. For each of these j, $\mathbf{H}[j,j] \ne 0$, implying $rank(\mathbf{H}) \ge \frac{N}{2t}$. □

10 Special Cases

10.1 Displacement Rank

The *displacement rank* of a matrix A with respect to matrices $\mathbf{L}, \mathbf{R}$ is defined as the rank of the *error matrix*

$$\mathbf{E} = \mathbf{LA} - \mathbf{AR}.$$

The concept of displacement rank has been used to generalize and unify common structured matrices such as Hankel, Toeplitz, Vandermonde, and Cauchy matrices; these matrices all have low displacement ranks with respect to diagonal or shift matrices being $\mathbf{L}$ and $\mathbf{R}$. Olshevsky and Shokrollahi [38] defined the confluent Cauchy-like matrices to be the class of matrices with low displacement rank with respect to Jordan form matrices; this class of matrices generalized and unified the previously mentioned common structured matrices. Our class of structured matrices captures the class of matrices with low displacement rank with respect to a more general form of matrices.

10.1.1 Basic conditions on L and R

Let $\mathbf{L}$ be Krylov efficient and suppose that we know its characteristic polynomial $c_{\mathbf{L}}(X)$. Let $\mathbf{R}$ be triangular (throughout this section, we will assume it is upper triangular) and Δ-banded and suppose that its eigenvalues are disjoint from **L**'s. We will show that the columns of **A** satisfies a standard **L**-matrix recurrence. Suppose $rank(\mathbf{E}) = r$; we can then express any column of **E** in terms of a basis $\mathbf{d}_0, \ldots, \mathbf{d}_{r-1}$. Let $\mathbf{f}_i = \mathbf{A}[:, i]$.

$$\begin{aligned}
\mathbf{L}\mathbf{f}_i - \sum_{j=i-\Delta}^{i} \mathbf{f}_j \mathbf{R}[j,i] &= \sum_{\ell=0}^{r-1} c_{i,\ell}\mathbf{d}_\ell \\
(\mathbf{L} - \mathbf{R}[i,i]\mathbf{I})\mathbf{f}_i &= \sum_{j=i-\Delta}^{i-1} \mathbf{f}_j \mathbf{R}[j,i] + \sum_{\ell=0}^{r-1} c_{i,\ell}\mathbf{d}_\ell \\
\mathbf{f}_i &= \sum_{j=i-\Delta}^{i-1} (\mathbf{L} - \mathbf{R}[i,i]\mathbf{I})^{-1}\mathbf{R}[j,i]\mathbf{f}_j + \sum_{\ell=0}^{r-1} (\mathbf{L} - \mathbf{R}[i,i]\mathbf{I})^{-1} c_{i,\ell}\mathbf{d}_\ell \\
\mathbf{f}_i &= \sum_{j=i-\Delta}^{i-1} \frac{\mathbf{R}[j,i]}{(X - \mathbf{R}[i,i])} \otimes \mathbf{f}_j + \sum_{\ell=0}^{r-1} \frac{c_{i,\ell}}{(X - \mathbf{R}[i,i])} \otimes \mathbf{d}_\ell
\end{aligned}$$

where $\otimes$ is $p \otimes \mathbf{f} = p(\mathbf{L})\mathbf{f}$. Note that the disjoint eigenvalue assumption asserts that $\mathbf{L} - \mathbf{R}[i,i]$ is invertible for all i, hence the $\mathbf{f}_i$ are well-defined and unique. The assumption of knowing $c_{\mathbf{L}}(X)$ is necessary since these recurrence coefficients are actually treated as elements of the quotient ring $\mathbb{F}[X]/(c_{\mathbf{L}}(X))$ and not $\mathbb{F}(X)$ (see Section 6.2).

This last equation says that $\mathbf{A}^T$ exactly satisfies a nice matrix recurrence with error, thus the algorithms of Section 6 and 7 allow us to compute **Ab** and $\mathbf{A}^T\mathbf{b}$.

We also note that the conditions on **L**, **R** are symmetric. Let (a) and (b) be matrix properties that apply through transposition (i.e. if **A** satisfies (a) then so does $\mathbf{A}^T$). Now suppose that **Ab** and $\mathbf{A}^T\mathbf{b}$ admit fast multiplication algorithms when **L** satisfies (a) and **R** satisfies (b). Then they are also efficient when **R** satisfies (a) and **L** satisfies (b). This is because $\mathbf{R}^T\mathbf{A}^T - \mathbf{A}^T\mathbf{L}^T = -\mathbf{E}^T$ is also rank r, let $\mathbf{L}' = \mathbf{R}^T$ and $\mathbf{R}' = \mathbf{L}^T$ which satisfy the appropriate properties so $\mathbf{A}^T$ and $(\mathbf{A}^T)^T$ admit fast multiplication.

Theorem 10.1. *Suppose* $rank(\mathbf{LA} - \mathbf{AR}) = r$ *for* **L** *that is (α, β)-Krylov efficient and* **R** *that is upper triangular and* Δ*-banded. Suppose furthermore that* **R** *and* **L** *have disjoint eigenvalues. Then we can compute* **Ab** *and* $\mathbf{A}^T\mathbf{b}$ *for any vector* **b** *in* $\tilde{O}(\Delta(\Delta + r + \beta)N)$ *with* $\tilde{O}((\Delta^{\alpha_r} + \Delta\alpha)N)$*. preprocessing.*

Proof. By Theorem 7.1 □

A Vandermonde-like matrix **V** is a matrix that has low displacement rank with respect to **L** = **D**, a diagonal matrix, and $\mathbf{R} = \mathbf{S}^T$. So the columns of **V** must follow the recurrence

$$\mathbf{f}_i = \frac{1}{X}\mathbf{f}_{i-1} + \frac{c_i}{X} \otimes \mathbf{d}$$

where $p(X) \otimes \mathbf{f} = p(\mathbf{D}) \otimes \mathbf{f}$.

As in the example of Vandermonde-like matrices, all previous displacement rank results in literature have **L** and **R** in Jordan normal form, all of which are captured by Olshovsky and Shokrollahi [38]. The results in this section cover all **L** and **R** in Jordan normal form that have distinct eigenvalues. We note that it is not possible to capture all matrices with low displacement rank with respect to **L**, **R** in Jordan normal form. In particular, any arbitrary matrix has low displacement rank with respect to **L** = **R** = **I**. However, there are fundamental results that have **L** and **R** with shared eigenvalues; for example, Toeplitz matrices have low displacement rank with respect to $\mathbf{L} = \mathbf{R} = \mathbf{S}^T$. We can generalize our results to capture these as well.

10.1.2 Further classes of displacement rank

The equation $\mathbf{LA} - \mathbf{AR} = \mathbf{E}$ only has a unique solution if **L** and **R** have disjoint eigenvalues. To see this, suppose **L** and **R** have at least one shared eigenvalue; we will find a non-zero solution to $\mathbf{LA} - \mathbf{AR} = 0$. Let $\mathbf{L} = \mathbf{X}\mathbf{J}_0\mathbf{X}^{-1}$ and $\mathbf{R} = \mathbf{Y}\mathbf{J}_1\mathbf{Y}^{-1}$ be the Jordan decompositions of the two matrices. We then have $\mathbf{J}_0\mathbf{X}^{-1}\mathbf{A}\mathbf{Y} - \mathbf{X}^{-1}\mathbf{A}\mathbf{Y}\mathbf{J}_1 = 0$. Since **X** and **Y** are full rank, a non-zero solution **A** exists if and only if there exists a nonzero solution **M** to $\mathbf{J}_0\mathbf{M} - \mathbf{M}\mathbf{J}_1 = 0$. Since **L** and **R** share an eigenvalue, there exists x and y such that $\mathbf{J}_0[x,x] = \mathbf{J}_1[y,y]$ and $\mathbf{J}_0[x,:], \mathbf{J}[:,y]$ are only non-zero on the diagonal. Then $\mathbf{M}[x,y]$ is completely free; the the $[x,y]$ element of $\mathbf{J}_0\mathbf{M}-\mathbf{M}\mathbf{J}_1$ is $\mathbf{M}[x,y](\mathbf{J}_0[x,x]-\mathbf{J}_1[y,y])=0$. Therefore, $\mathbf{LA}-\mathbf{AR}=\mathbf{E}$ has multiple solutions if $\mathbf{L}$ and $\mathbf{R}$ have shared eigenvalues.

As such, to extend our results beyond disjoint eigenvalues, we need to allow for extra parameters to specify $\mathbf{A}$. In this section, we deal with cases where the extra parameters can be provided by the initial conditions of our recurrence. As an example, we rework our derivation from the previous section with $\mathbf{L}=\mathbf{R}=\mathbf{S}^T$, which corresponds to Toeplitz-like matrices. We have $\mathbf{S}^T\mathbf{f}_i-\mathbf{f}_{i-1}=\sum_{\ell=0}^{r-1}c_{i,\ell}\mathbf{d}_\ell$. This gives us the recurrence $\mathbf{f}_{i-1}=X\otimes\mathbf{f}_i-\sum_{\ell=0}^{r-1}c_{i,\ell}\mathbf{d}_\ell$, which means we can apply our matrix vector multiplication algorithm to $\mathbf{A}$ and $\mathbf{A}^T$. This example illustrates a simple technique to expand the class of $\mathbf{L},\mathbf{R}$ that we capture: we can simply put a different element of the sequence on the left side of the recurrence equation. Also note that the free element of $\mathbf{A}$ is specified by the initialization $\mathbf{f}_0$.

If we define a matrix $\mathbf{M}=\mathbf{R}-X\mathbf{I}$ over $\mathbf{F}[X]^{N\times N}$, then we can express the relation among the $\mathbf{f}_i$ with

$$\sum_{j=0}^{N}\mathbf{M}[j,i]\otimes f_j=\sum_{\ell=0}^{r-1}c_{i,\ell}\mathbf{d}_\ell \tag{28}$$

We would like to extract a Δ-width $\mathbf{R}$-recurrence (with error) from this relationship. This requires specifying some ordering of the $\mathbf{f}_i$ such that the relation above can be interpreted as a Δ-width recurrence. Equivalently, we need to have permutations σ,τ such that equation (28) reduces to the recurrence

$$\mathbf{M}[\sigma(i),\tau(i)]\otimes\mathbf{f}_{\sigma(i)}=\sum_{j=1}^{\Delta}\mathbf{M}[\sigma(i-j),\tau(i)]\otimes\mathbf{f}_{\sigma(i-j)}+\sum_{\ell=0}^{r-1}c_{\tau(i),\ell}\mathbf{d}_\ell \tag{29}$$

Given a permutation σ, define a permutation matrix $\mathbf{P}(\sigma)$ such that $\mathbf{P}[i,\sigma(i)]=1$ and 0 otherwise. Equation (28) will only reduce to recurrence (29) if $\mathbf{P}(\sigma)\mathbf{M}\mathbf{P}(\tau)$ is upper triangular and Δ-banded in rows $[\Delta+1:N-1]$. In other words, for $i>\Delta$, $\mathbf{M}[\sigma(i),\tau(j)]\neq 0$ if and only if $i\le j\le i+\Delta$. Furthermore, the rational recurrence is only usable if the polynomial $\mathbf{M}[\sigma(i),\tau(i)]$ evaluated at $\mathbf{L}$ results in an invertible matrix for $i>\Delta$. In this case, we will say $\mathbf{M}$ is *pseudo similar*to a *pseudo Δ-banded upper triangular matrix.*

Going back to the example of $\mathbf{L}=\mathbf{R}=\mathbf{S}^T$, $\mathbf{M}$ is a two-banded matrix with X on the diagonal and 1 on the superdiagonal. Define $\sigma(i)=N-i$ and $\tau(i)=i+1$ (with $\tau(N-1)=0$). Then $\mathbf{P}(\sigma)\mathbf{M}\mathbf{P}(\sigma)\mathbf{P}(\tau)$ is a two-banded matrix with 1 on the diagonal and X on the super diagonal (ignoring the first column). With these permutations, recurrence (29) is equivalent to the recurrence derived earlier: $\mathbf{f}_{i-1}=X\otimes f_i-\sum_{\ell=0}^{r-1}c_{i,\ell}\mathbf{d}_\ell$.

By applying our previously established bounds to this recurrence, we can derive an explicit runtime. Note that our computation will involve both the matrices $\mathbf{L}$ and $\mathbf{R}$.

Theorem 10.2. *Suppose $\mathbf{M}(\mathbf{L},\mathbf{R})$ is pseudo similar to a pseudo Δ-banded upper triangular matrix and $\mathbf{R}$ is (α,β)-Krylov efficient. Let $r=rank(\mathbf{LA}-\mathbf{AR})$ for some matrix $\mathbf{A}$. Then with $\tilde{O}((\Delta^{\alpha_r}+\Delta\alpha)N)$ pre-processing, both $\mathbf{Ab}$ and $\mathbf{A}^T\mathbf{b}$ can be computed, for any $\mathbf{b}$, in $\tilde{O}(\Delta(\Delta+r+\beta)N)$ operations.*

The techniques in this section recover all of the classic Displacement Rank results: fast matrix vector multiplication for Cauchy-like, Vandermonde-like, Toeplitz-like, and Hankel-like matrices. In addition, as illustrated in the example for Toeplitz-like matrices, the permutation techniques of this section allows for any $\mathbf{L}$ and $\mathbf{R}$ that are Jordan blocks. Note that for all of these classes $\Delta,\alpha,\beta=O(1)$, giving us an $\tilde{O}(rN)$ runtime.

Olshevsky and Shokrollahi discuss $\mathbf{L}$ and $\mathbf{R}$ in general Jordan normal form [38]. Unfortunately, they do not explicitly deal with multiple solutions to the displacement rank equation; they do claim an extension of their results could handle the general case. Similarly, for our recurrence-based approach, if we extend our rational recurrences to use pseudo-inverses and allow for extra parameters specifying $\mathbf{A}$ to be added to the errors $\mathbf{d}$, we could be able to handle $\mathbf{L},\mathbf{R}$ sharing arbitrary eigenvalues. We leave this for future investigation.

10.2 Orthogonal Polynomials

Driscoll, Healy and Rockmore [17] found a superfast matrix vector multiplication algorithm for matrices whose rows are orthogonal polynomials. They take advantage of the three-term recurrence that any family of orthogonal polynomials must satisfy. This recurrence also implies that our results capture theirs.

In addition to being more general, we view our algorithm as being simpler and more direct. In fact, there are two ways in which the Driscoll et al. algorithm can be viewed as a direct application of our algorithm. The Driscoll et al. algorithm actually computes the projection of a vector **b** onto the orthogonal polynomials $p_0(X), \ldots, p_{N-1}(X)$ as defined by

$$\hat{\mathbf{b}}[i] = \sum_{j=0}^{N-1} b_j p_i(z_j).$$

Note that the algorithm combines the matrix multiplication with a multi-point evaluation, while our algorithm more directly operates on the polynomials themselves instead of their evaluations. Thus if **A** is the matrix containing the orthogonal polynomials in its rows, then $\hat{\mathbf{b}} = (\mathbf{A}\mathbf{V}^T)\mathbf{b}$, where **V** is the Vandermonde matrix with evaluation points $z_0, \ldots, z_{N-1}$. In fact, it can be shown that the operations performed by the Driscoll et. al. algorithm are actually equivalent to first computing the projections onto monomials $\mathbf{V}^T\mathbf{b}$, and then applying our algorithm to multiply by **A**. The second way to use our algorithm to compute the result is by interpreting the orthogonal polynomial evaluations as a matrix recurrence (5) with $\mathbf{R} = \operatorname{diag}\langle z_0, \ldots, z_{N-1}\rangle, \mathbf{f}_0 = \mathbf{1}$. This is a more general method than the $\mathbf{A}\mathbf{V}^T$ factorization because it does not depend on the functions being polynomials - and when they are, it turns out to be equivalent because $\mathcal{K}(\mathbf{R}, \mathbf{f}_0) = \mathbf{V}$.

Orthogonal polynomials are also much more structured than our generalized polynomials, which we exemplify here by providing a simple algorithm for matrix-vector multiplication involving inverses. Suppose we have orthogonal polynomials $p_1(X), \ldots, p_N(X)$ and a matrix **A** such that $p_i(X) = \sum_{j=0}^{N-1} \mathbf{A}[i,j]X^j$. By definition [14], $\int p_i(X)\bar{p_j}(X)d\mu(X) = \delta_{ij}$ for some measure $\mu(X)$. Define the moment matrix $\mathbf{M}[i,j] = \int \bar{X}^j X^i d\mu(X)$. Note that if the measure is supported on the real line, then $\mathbf{M}[i,j]$ is a function of $i+j$ and **M** is Hankel. Similarly, if μ is supported on the complex circle, $\mathbf{M}[i,j]$ is a function of $i-j$ and **M** is Toeplitz. Note however that **M** may not have special structure in general.

Theorem 10.3. *If* **A** *is a matrix of orthogonal polynomials with respect to the measure* $\mu(X)$*, and* **M** *is the moment matrix over the measure, then*

$$\mathbf{AMA}^* = \mathbf{I}$$

Proof. Consider the following sequence of relations for $0 \le i, j < N$:

$$\begin{aligned}
(\mathbf{AMA}^*)[i,\ell] &= \sum_{j=0}^{N-1} \mathbf{A}[i,j] \sum_{k=0}^{N-1} \mathbf{M}[j,k]\mathbf{A}^*[k,\ell] \\
&= \sum_{j=0}^{N-1}\sum_{k=0}^{N-1} \mathbf{A}[i,j]\left(\int X^j \overline{X}^k d\mu(X)\right)\mathbf{A}^*[k,\ell] \\
&= \int \left(\sum_{j=0}^{N-1} \mathbf{A}[i,j]X^j\right)\left(\sum_{k=0}^{N-1} \mathbf{A}^*[k,\ell]\overline{X}^k\right) d\mu(X) \\
&= \int p_i(X)\overline{p}_\ell(X) d\mu(X) \\
&= \delta_{i,\ell},
\end{aligned}$$

as desired. □

In the case that our measure is supported on the real line, $\mathbf{A}^* = \mathbf{A}^T$ and **M** is Hankel. Note that classical superfast vector multiplication algorithms for Hankel matrices and their inverses have been well established [22]. This result implies that for orthogonal polynomials, an algorithm for the $\mathbf{A}^T\mathbf{b}$ implies an algorithm for $\mathbf{A}^{-1}\mathbf{b}$ and an algorithm for $\mathbf{Ab}$ implies an algorithm for $\mathbf{A}^{-T}\mathbf{b}$. So our algorithms presented in Sections 4 and 6 immediately imply algorithms for inverses in the case that our matrices contain orthogonal polynomials.

We observe that our more general class of polynomials do not follow this nice structure. $\mathbf{A}^{-1}\mathbf{A}^{-T}$ may not be a Hankel matrix in general; it may not even be symmetric.

10.3 Constant Transition Matrix

There is a great body of research relating to linear recursive sequences in which the recurrence has a constant transition matrix [35]. These sequences can be generalized from scalars to polynomials, which leads to a recurrence in which the $g_{i,j}(X)$ are independent of i. Some well-known families of polynomials fall under this setting, most prominently the various types of Chebyshev polynomials which all satisfy a width-1 recurrence with constant transitions [21]. We also note that with some transformations, other families such as the Bernoulli numbers fall under this category of recurrence, which we discuss in the next subsection.

In this case, $\mathbf{T}_i$ as defined by Section 3 are all identical. We can then express $\mathbf{f}_i = \mathbf{T}^{i-t}\mathbf{f}$ where $\mathbf{f} = \begin{bmatrix}\mathbf{f}_0 & \cdots & \mathbf{f}_t\end{bmatrix}^T$. In this case, we have

$$\mathbf{A}^T\mathbf{b} = \sum_{i=0}^{t} \mathbf{b}[i]\mathbf{f}_i + \sum_{i=t+1}^{N} \mathbf{b}[i]\mathbf{T}^{i-t}\mathbf{f}.$$

Similarly, we have

$$(\mathbf{Ab})[i] = \mathbf{b}^T\mathbf{T}^{i-t}\mathbf{f}.$$

We note that $\mathbf{T}$ is essentially a traditional companion matrix of a linear recursive sequence [27], except that the coefficients in $\mathbf{T}$ are polynomials instead of constants.

With this constant transition matrix, we can use the approach of Fiduccia [19] to reduce both the pre-processing and multiplication time to $\tilde{O}(Nt)$. Fiduccia investigates linear recurrences, and uses a fact about companion matrices to derive a $O(t\log t\log n)$ method to find any nth element of a linear recurrence with t terms. Specifically, given a companion matrix $\mathbf{C} \in \mathscr{R}^{n\times n}$ with characteristic polynomial $p(Y)$, the transformations of multiplication by $\mathbf{C}$ in $\mathscr{R}^n$ is isomorphic to multiplication by Y in $\mathscr{R}[Y]/(p(Y))$. Therefore computing $\mathbf{C}^k$ can be reduced to computing $Y^k \pmod{p(Y)}$.

We can generalize this approach to our recurrence, the only change being that the elements of our companion matrix $\mathbf{T}$ are polynomials - specifically, they are the $g_{i,j}(X)$ of (10) (which now don't depend on i, so we will denote them $g_j(X)$). Then we can define the characteristic polynomial $p(X,Y) = Y^{t+1} - \sum_{j=0}^{t} g_j(X)Y^{t-j}$ of $\mathbf{T}$. The bottlenecks of Algorithm 1 and Algorithm 2 lie in the pre-computations and multiplications of the ranged transition matrices (see Lemmas 4.4 and 6.5), so we focus our attention on them. In this case, we need to examine $\mathbf{T}^{2^b}$ for $b \in [0, \log_2 N]$. As noted, the following formulations of jumping the recurrence by i rows are equivalent because of the isomorphism between $\left(\mathbf{T}, \mathbb{F}[X]^{t+1}\right)$ and $\left(Y, \mathbb{F}[X][Y]/(p(X,Y))\right)$:

$$\begin{bmatrix} f_{k+i}(X) \\ \vdots \\ f_{k+i+t}(X) \end{bmatrix} = \mathbf{T}^i \begin{bmatrix} f_k(X) \\ \vdots \\ f_{k+t}(X) \end{bmatrix}$$

$$f_{k+i}(X) + \cdots + f_{k+i+t}(X)Y^t = Y^i(f_k(X) + \cdots + f_{k+t}(X)Y^t) \pmod{p(X,Y)} \tag{30}$$

For the preprocessing step, we will calculate polynomials $h_i(X,Y) : i = 2^b$ such that $\deg_Y(g_i) \le t$ and $h_i(X,Y) = Y^i \pmod{p(X,Y)}$; these play the role of $\mathbf{T}^i$. Note that reducing a polynomial's Y-degree $\pmod{p(X,Y)}$ does not increase its overall degree:

Lemma 10.4. *Let* $q(X,Y), r(X,Y) \in \mathbb{F}[X,Y]$ *such that* $q(X,Y) \equiv r(X,Y) \pmod{p(X,Y)}$, $\deg(q) \le d$, *and* $\deg_Y(r) < \deg(p) = t+1$. *Then* $\deg(r) \le d$.

Proof. Consider the following process of computing $r(X,Y)$ starting from $q(X,Y)$; at every iteration, we will find an equivalent polynomial $\pmod{p(X,Y)}$ but of lower Y-degree. If the polynomial has Y-degree at most t, we are done and it must be $r(X,Y)$. Otherwise, replace its term of highest Y-degree by substituting

$$Y^n = Y^{n-t-1}Y^{t+1} = Y^{n-t-1}\sum_{j=0}^{t} g_j(X)Y^{t-j},$$

which does not change its value $\pmod{p(X,Y)}$. It also cannot raise the polynomial's degree because of the degree bound $\deg(g_j(X)) \le j+1$. □

In particular, $\deg(h_i(X,Y)) \le i$. Given $h_i(X,Y)$, we can compute $h_{2i}(X,Y)$ by squaring $h_i(X,Y)$ and reducing (mod $p(X,Y)$). Note that $\deg_X(h_i) \le i$ and $\deg_Y(h_i) \le t$. Treating h_i as a polynomial in Y, squaring it takes $\widetilde{O}(t)$ operations over its coefficients, which are elements of $\mathbb{F}[X]$ [12]. This in turn takes $\widetilde{O}(i)$ operations over $\mathbb{F}$. Furthermore, computing the polynomial remainder of $h_{2i}(X,Y)$ by $p(X,Y)$ has the same complexity [13]. Thus each $h_i(X,Y)$ takes $\widetilde{O}(ti)$ operations to compute, and calculating $h_1,\dots,h_N$ through repeated squaring takes $\widetilde{O}(tN)$ operations in total.

Now suppose we are given the pre-computations and are performing a matrix multiplication. For concreteness, we will examine the top level of the recurrence for $\mathbf{A}^T\mathbf{b}$. As noted in Lemma 4.4, the bottleneck was performing a matrix-vector multiplication by $\mathbf{T}_{[0:N/2]}$, which was done in $\widetilde{O}(t^2N)$ operations. In this constant-transition case, the multiplication can be performed via (30), where we can replace Y^i with the pre-computed $h_i(X,Y)$. This is again a multivariate polynomial multiplication and reduction, where the degree in X is $O(N)$ and the degree in Y is $O(t)$. Thus only $\widetilde{O}(tN)$ operations are required to perform the ranged-transition multiplication. Replacing this cost in the runtime analysis of Lemma 4.4 implies that the whole algorithm only needs $\widetilde{O}(tN)$ operations.

The same analysis applies for computing $\mathbf{Ab}$ by modifying Lemma 6.5. Finally, we incur the standard Krylov efficiency cost in the case of matrix recurrences. Thus, we have shown that

Theorem 10.5. *For any nice recurrence as in* (10) *where $g_{i,j}(X)$ are independent of i, with $\widetilde{O}(tN)$ pre-processing operations, any $\mathbf{A}^T\mathbf{b}$ and $\mathbf{Ab}$ can be computed with $\widetilde{O}(tN)$ operations over $\mathbb{F}$. For an $\mathbf{R}$-matrix recurrence* (5) *that is nice and $\mathbf{R}$ is (α,β)-Krylov efficient, pre-processing and computation need additional $\tilde{O}(t\alpha N)$ and $\tilde{O}(t\beta N)$ operations respectively.*

10.4 Bernoulli Polynomials

Bernoulli polynomials appear in various topics of mathematics including the Euler-Maclaurin formulae relating sums to integrals, and formulations of the Riemann zeta function [8]. In this section we will demonstrate how our techniques are flexible enough to calculate quantities such as the Bernoulli numbers, even though they do not ostensibly fall into the framework of bounded-width linear recurrences.

Bernoulli polynomials are traditionally defined by the recursive formula

$$B_i(X) = X^i - \sum_{k=0}^{i-1} \binom{i}{k} \frac{B_k(X)}{i-k+1}$$

with $B_0(X) = 1$ [6]. This recurrence seemingly cannot be captured by our model of recurrences, since each polynomial depends on *every previous polynomial.* However, with some additional work, a recurrence with bounded recurrence width can also capture this larger recurrence, thereby facilitating superfast matrix-vector multiplication involving B.

We start out by computing $B_i(0)$ for each i. For notational convenience, we may drop the 0 and use B_i to denote $B_i(0)$. In particular

$$B_i = \sum_{k=0}^{i} (-1)^k \frac{k!}{k+1} \left\{ {i \atop k} \right\}$$

where $\left\{ {i \atop k} \right\}$ denotes the Stirling numbers of the second kind [6]. To compute B_i for $0 \le i \le N$, we define the Stirling matrix $\mathbf{W}$ such that $\mathbf{W}[i,j] = \left\{ {i \atop j} \right\}$ and a vector a diagonal matrix $\mathbf{D}$ such that $\mathbf{D}[i,i] = i!$, and a vector $\mathbf{x}$ such that $\mathbf{x}[k] = \frac{(-1)^k k!}{k+1}$; the vector $\mathbf{b} = \mathbf{Wx}$ will contain $B_i(0)$ as $\mathbf{b}[i]$.

Lemma 10.6. *If $\mathbf{W}[i,j] = \left\{ {i \atop j} \right\}$, we can multiply $\mathbf{Wx}$ or $\mathbf{W}^T\mathbf{x}$ for any vector $\mathbf{x}$ with $O(N\log^2 N)$ operations.*

Proof. Note that the Stirling numbers satisfy the recurrence [6]

$$\left\{ {i+1 \atop k} \right\} = k \left\{ {i \atop k} \right\} + \left\{ {i \atop k-1} \right\}.$$

If we let $\mathbf{f}_i = \mathbf{W}[i,:]^T$, the recurrence gives us that $\mathbf{f}_{i+1} = (\mathbf{D}+\mathbf{S})\mathbf{f}_i$. Note that by Theorem 8.1, $(\mathbf{D}+\mathbf{S})$ is $(1,1)$-Krylov efficient. Theorems 6.8 and 4.5 complete the proof. □

Corollary 10.7. *We can compute $B_i(0)$ for all i with $O(N\log^2 N)$ operations.*

We note that the $B_i(0)$ are actually the Bernoulli numbers, and our algorithm facilitates the computation of the Bernoulli numbers in the same runtime as recent state-of-the-art ad hoc approaches [25]. (See Section 10.6 for more.)

We now focus on a matrix $\mathbf{Z}$ such that $\mathbf{Z}[i,j] = B_i(\alpha_j)$ for $\alpha_0,\dots,\alpha_{N-1} \in \mathbb{F}$. Note that a superfast multiplication algorithm for $\mathbf{Z}$ immediately implies one for $\mathbf{B}$ since $\mathbf{Z} = \mathbf{B}\mathbf{V}_\alpha^T$, where $\mathbf{V}_\alpha$ is the Vandermonde matrix defined by evaluation points $\alpha_0,\dots,\alpha_{N-1}$. We take advantage of the following identity relating $B_i(X)$ to B_i:

$$B_{i+1}(X) = B_{i+1} + \sum_{k=0}^{i} \frac{i+1}{k+1} \left\{ {i \atop k} \right\} (X)_{k+1}$$

where $(X)_{k+1} = X(X-1)\cdots(X-k)$ denotes the falling factorial [6].

Lemma 10.8. *Suppose a matrix $\mathbf{F}$ is defined such that $f_i(X) = \sum_{j=0}^{N-1} \mathbf{F}[i,j]X^j = (X)_i$. Then for any vector $\mathbf{x}$, we can multiply $\mathbf{F}\mathbf{x}$ and $\mathbf{F}^T\mathbf{x}$ in $O(N\log^2 N)$.*

Proof. By definition $f_{i+1}(X) = (X-i)f_i(X)$, and hence, Theorems 6.8 and 4.5 imply that we can multiply $\mathbf{F}$ and $\mathbf{F}^T$ by vectors in $O(N\log^2 N)$. □

Theorem 10.9. *For any vector $\mathbf{b}$, we can compute $\mathbf{Z}\mathbf{b}$ or $\mathbf{Z}^T\mathbf{b}$ with $O(N\log^2 N)$ operations.*

Proof. Suppose we define $\mathbf{C}_{i+1,j} = \frac{i+1}{j+1}\left\{ {i \atop j} \right\}$. Note that $\mathbf{C}$ is just the Stirling matrix $\mathbf{W}$ multiplied by two diagonal matrices, implying it supports $O(N\log^2 N)$ vector multiplication. Furthermore, as in our previous lemma, define $\mathbf{F}$ to be a matrix corresponding to the falling factorials. Then $(\mathbf{C}\mathbf{V}_\alpha\mathbf{F}^T)[i,j] = B_i(\alpha_j) - B_i$. So if we define a matrix $\mathbf{M}$ such that $\mathbf{M}[i,j] = B_i$, $\mathbf{Z} = \mathbf{C}\mathbf{V}_\alpha\mathbf{F}^T + \mathbf{M}$. Thus multiplying by $\mathbf{Z}$ or $\mathbf{Z}^T$ reduces to multiplying by these components, which by Lemmas 10.6 and 10.8 can be done in $O(N\log^2 N)$ operations. □

As discussed previously, this theorem immediately implies that we can compute matrix-vector multiplications involving $\mathbf{B}$ in $O(N\log^2 N)$ operations as well. We note that the numerical errors that may arise throughout these computations come from rounding and not having enough bits to represent the data; there are no catastrophic cancellation errors. As such, we only need to analyze the bit complexity of the algorithm to understand its numerical properties (which we do in Section 10.6).

10.5 Efficient Jordan Decomposition

Every linear operator admits a Jordan normal form, which in some senses is the smallest and most uniform representation of the operator under any basis. Knowing the Jordan decomposition of a matrix permits many useful applications. For example, being able to quickly describe and manipulate the change of basis matrix, which is a set of generalized eigenvectors, subsumes calculating and operating on the eigenvectors of the matrix.

Another use of the Jordan decomposition is being able to understand the evaluation of any analytic function on the original matrix. We highlight this application because of its connection to Krylov efficiency in Section 8, which can be solved by evaluating the matrix at a certain polynomial function. Computing matrix functions has widespread uses, and there is a significant body of research on the design and analysis of algorithms for various matrix functions, including the logarithm, matrix sign, pth root, sine and cosine [26]. Perhaps the most prominent example of a matrix function is the matrix exponential, which is tied to the theory of differential equations - systems are well-approximated around equilibria by a linearization $\mathbf{x}'(t) = \mathbf{M}\mathbf{x}(t)$, which is solved by the exponential $\mathbf{x}(t) = e^{t\mathbf{M}}\mathbf{x}(0)$ [24]. We note that much work has been done on computing the matrix exponential in particular without going through the Jordan decomposition, since it is generally hard to compute [41].

We will show how our techniques facilitate fast computation and succinct description of the Jordan decomposition for certain classes of matrices, and recapitulate how it can be used to easily compute matrix functions.

Consider a matrix $\mathbf{M}$ with a Jordan decomposition $\mathbf{M} = \mathbf{A}\mathbf{J}\mathbf{A}^{-1}$. We say that $\mathbf{M}$ is (α,β)-Jordan efficient if $\mathbf{A}$ and $\mathbf{A}^{-1}$ admit super-fast matrix-vector multiplication in $\tilde{O}(\beta N)$ operations with $\tilde{O}(\alpha N)$ pre-processing steps.

Such matrices satisfy the property that their evaluation at any analytic function f also admits super-fast matrix-vector multiplication.

Lemma 10.10 ([26]). *Let* $\mathbf{J}$ *be a Jordan block with diagonal* λ *and* $f(z)$ *be a function that is analytic at* λ*. Then*

$$f(\mathbf{J}) = \begin{pmatrix} f(\lambda) & f'(\lambda) & \cdots & \frac{f^{(n-1)}(\lambda)}{(n-1)!} \\ 0 & f(\lambda) & \cdots & \frac{f^{(n-2)}(\lambda)}{(n-2)!} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & f(\lambda) \end{pmatrix}$$

Proof. The proof follows from considering the Taylor series expansion $f(z) = \sum \frac{f^{(i)}(\lambda)}{i!}(z-\lambda)^i$ and plugging in $\mathbf{J} = \lambda\mathbf{I} + \mathbf{S}^T$. □

Lemma 10.11. *Let* $\mathbf{M}$ *be* (α,β)*-Jordan efficient and* $f(z)$ *be a function that is analytic at the eigenvalues of* $\mathbf{M}$*. Then* $f(\mathbf{M})$ *admits super-fast matrix multiplication in* $\tilde{O}(\beta N)$ *operations, with* $\tilde{O}(\alpha N)$ *pre-processing.*

Proof. We can write a matrix-vector product as

$$f(\mathbf{M})\mathbf{b} = f(\mathbf{AJA}^{-1})\mathbf{b} = \mathbf{A}f(\mathbf{J})\mathbf{A}^{-1}\mathbf{b}$$

By Lemma 10.10, assuming access to the multi-point Hermite-type evaluations of f at the eigenvalues of $\mathbf{J}$, the product $f(\mathbf{J})\mathbf{b}$ is a series of Toeplitz multiplications which can be computed in time $O(N\log N)$. Thus this product is bottlenecked by the time it takes to multiply by $\mathbf{A}$ and $\mathbf{A}^{-1}$, and the result follows. □

Here are some cases of Jordan-efficient matrices:

1. Suppose that $\mathbf{M}$ is a Jacobi matrix. Then it is diagonalizable with $\mathbf{M} = \mathbf{ADA}^{-1}$, and $\mathbf{A}$ is an orthogonal polynomial matrix. In this case, multiplication by $\mathbf{A}$ is our standard result and multiplication by $\mathbf{A}^{-1}$ can be done using some properties of orthogonal polynomials, described in section 10.2.

2. Suppose that $\mathbf{M}$ is triangular Δ-banded, and $\mathbf{M}$'s minimal polynomial equals its characteristic polynomial. In this case, it turns out that the change of basis matrix can be expressed as $\mathbf{AV}^T\mathbf{P}$, where $\mathbf{A}$ is a Δ-width recurrence, $\mathbf{V}$ is a confluent Vandermonde matrix, and $\mathbf{P}$ is a permutation matrix, and furthermore $\mathbf{AV}^T$ is triangular. We can show how to describe $\mathbf{A}$ with $\widetilde{O}(\Delta^\omega N)$ operations, as well as perform matrix-vector multiplication by $\mathbf{A}$ and $\mathbf{A}^{-1}$ in $\widetilde{O}(\Delta^2 N)$ operations. The full proof is detailed below.

3. When $\mathbf{M}$ is triangular Δ-banded and diagonal, we can also compute and multiply by $\mathbf{A}$ efficiently using similar techniques to the above case.

Now consider a triangular Δ-banded $N\times N$ matrix $\mathbf{M}$ whose minimal polynomial has full degree. Throughout this section, we assume $\mathbf{M}$ is upper triangular. We will prove that $\mathbf{M}$ is $(\Delta^{\omega_{\mathcal{R}}}, \Delta^2)$-Jordan efficient in this case.

For such a matrix $\mathbf{M}$, we will use $c_{\mathbf{M}}(X)$ to denote the minimal and characteristic polynomial of $\mathbf{M}$. Define $\mathbf{F}$ to be the transpose of the companion matrix of $c_{\mathbf{M}}(X)$.

10.5.1 Recurrence of A

$\mathbf{M}$ is similar to $\mathbf{F_M}$ where $\mathbf{F_M}^T$ is the Frobenius companion matrix for $c_{\mathbf{M}}(X)$. In particular, suppose $c_{\mathbf{M}}(X) = X^N - c_{N-1}X^{N-1}\cdots - c_0$, then $\mathbf{F_M}$ will be

$$\begin{bmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \\ c_0 & c_1 & c_2 & \dots & c_{N-1} \end{bmatrix}$$

This matrix $\mathbf{F_M}^T$ is simply the Frobenius normal form, i.e., the canonical rational form, of $\mathbf{M}^T$.

For a matrix $\mathbf{A}$, we define $a_i(X)$ as the polynomial corresponding to the i^{th} row of $\mathbf{A}$, i.e., $a_i(X) = \sum_{j=0}^{N} \mathbf{A}[i,j]X^j$.

Lemma 10.12. *For some matrix* $\mathbf{A}$, $\mathbf{MA} = \mathbf{AF_M}$ *if and only if* $(\mathbf{M} - X\mathbf{I})\begin{bmatrix} a_0(X) \\ \vdots \\ a_{N-1}(X) \end{bmatrix} = 0 \mod c_{\mathbf{M}}(X)$.

Proof. Multiplying a vector by $\mathbf{F_M}$ in $\mathscr{R}$ is isomorphic to multiplying the corresponding polynomial by X in $\mathscr{R}[X]/(c_{\mathbf{M}}(X))$. This implies that $\mathbf{M}\begin{bmatrix} a_0(X) \\ \vdots \\ a_{N-1}(X) \end{bmatrix} = X\begin{bmatrix} a_0(X) \\ \vdots \\ a_{N-1}(X) \end{bmatrix} \pmod{c_{\mathbf{M}}(X)}$. □

Corollary 10.13. *There exists a* $\Delta - 1$ *width matrix* $\mathbf{A}$ *such that* $\mathbf{MA} = \mathbf{AF_M}$. *More specifically, the rows of* $\mathbf{A}$ *satisfy*

$$(X - \mathbf{M}[i,i])a_i(X) = \sum_{j=1}^{\min(\Delta, N-i)} \mathbf{M}[i, i+j]\, a_{i+j}(X) + d_i c_{\mathbf{M}}(X) \tag{31}$$

where $d_i \in \mathbb{F}$.

Proof. Note that the matrix $\mathbf{M}$ we are interested in is an upper-triangular, Δ-banded matrix. Our previous lemma implies that that $\mathbf{MA} = \mathbf{AF_M}$ if and only if $(\mathbf{M} - X\mathbf{I})\begin{bmatrix} a_0(X) \\ \vdots \\ a_{N-1}(X) \end{bmatrix} = 0 \pmod{c_{\mathbf{M}}(X)}$.

We remove the modulus and treat the $a_i(X)$ as polynomials over $\mathbb{F}[X]$ of degree less than N. The above condition becomes the system of equations 31. Since the left side has degree N, the d_i must have degree 0 so they are scalars. Thus if a family of polynomials is defined to satisfy the recurrence above, the associated $\Delta - 1$ width matrix A must satisfy $\mathbf{MA} = \mathbf{AF_M}$. □

We say that a recurrence of form (31) has *size N* if the total length of the recurrence has length N and $c(X)$ has degree N, so that all polynomials are bounded by degree $N-1$. We call the scalars $\mathbf{M}[\cdot,\cdot]$ the recurrence coefficients, and the d_i the error coefficients.

From now on, our use of $\mathbf{A}$ will denote such a $\Delta - 1$ width matrix. Note that independent of the d_i coefficients, the following divisibility lemma holds.

Lemma 10.14. $\prod_{j=0}^{i-1}(X - \mathbf{M}[j,j]) | a_i(X)$ *for* $0 \le i \le N-1$.

Proof. Proof by induction. As a base case, this is true for $a_{N-1}(X)$ by equation (31) since $\prod_{j=0}^{N-1}(x - \mathbf{M}[j,j]) = c_{\mathbf{M}}(X)$. And (31) gives us the inductive step. □

10.5.2 Conditions on the Error Coefficients

We showed in Section 10.5.1 that the equation $\mathbf{MA} = \mathbf{AF}$ is equivalent to a recurrence (31). In order to complete the task of finding $\mathbf{A}$ satisfying $\mathbf{M} = \mathbf{AFA}^{-1}$, we must find scalars d_i in the recurrence that lead to an invertible matrix $\mathbf{A}$. The goal of this subsection is in proving a strong sufficiency condition on such sequences d_i.

We will first show some equivalent conditions to $\mathbf{A}$ being invertible.

Definition 10.15. Given a set of points $p_0, \ldots, p_{N-1}$, let $n_i = \sum_{j=0}^{i-1} \mathbb{1}(p_j = p_i)$ be the number of preceding points identical to p_i. Then the confluent Vandermonde matrix, denoted $\mathbf{V}_{p_0,\ldots,p_{N-1}}$, is defined such that

$$\mathbf{V}[i,j] = \begin{cases} 0, & j < n_i \\ \frac{j!}{(j-n_i)!(n_i)!} {p_i}^{j-n_i}, & j \ge n_i \end{cases}$$

for $0 \le i, j \le N-1$. [8]

[8]Some sources define the confluent Vandermonde matrix with the factor of $n_i!$ in the denominator [39], and others do not. It makes no real difference, but is slightly more convenient for us to use the former notation.

Note that for any vector $\mathbf{y}$, $\mathbf{V}_{p_0,\dots,p_{N-1}}\mathbf{y}$ is equivalent to evaluating the polynomial $v(X) = \sum_{i=0}^{N-1} \mathbf{y}[i]X^i$ at $v^{(n_i)}(p_i)$ for each i. We define $\mathbf{V}_\mathbf{M}$ to be the confluent Vandermonde matrix $\mathbf{V}_{\mathbf{M}[0,0],\dots,\mathbf{M}[N-1,N-1]}$.

Lemma 10.16. *The following are true for* $\mathbf{A}$ *defined by recurrence (31), for any* d_i*.*

(a) $\mathbf{A}\mathbf{V}_\mathbf{M}^T$ *is upper triangular.*

(b) $a_i^{(n_j)}(\mathbf{M}[j,j]) = 0$ *for all* i *and* $j < i$*.*

(c) $\prod_{j<i}(X - \mathbf{M}[j,j]) \mid a_i(X)$ *for all* i*.*

Proof. We show the equivalence of the conditions. This is sufficient since (c) is true by Lemma 10.14.

$(a) \iff (b)$ As noted above, $\mathbf{A}\mathbf{V}_\mathbf{M}^T[i,j]$ can be understood as a Hermetian evaluations (evaluation a polynomial and its derivatives) and is equal to $a_i^{(n_j)}(\mathbf{M}[j,j])$. The equivalence follows since upper triangularity the same as saying $\mathbf{A}\mathbf{V}_\mathbf{M}^T[i,j] = 0$ for all $j < i$.

$(b) \iff (c)$ Fix an i. For every λ, let n_λ be its multiplicity in $\prod_{j<i}(X - \mathbf{M}[j,j])$. Note that condition (c) is equivalent to saying $(X-\lambda)^{n_\lambda} \mid a_i(X)$ for all λ. For a fixed λ, the divisibility condition $(X-\lambda)^{n_\lambda} \mid a_i(X)$ is equivalent to $a_i^{(k)}(\lambda) = 0$ for $k < n_\lambda$. This can be seen, for example, by considering the Taylor expansion of a_i around λ. Finally, the union of these equations over all λ is exactly (b), completing the equivalence.

□

The equivalence of the following conditions follows easily from the same reasoning.

Corollary 10.17. *The following are equivalent for* $\mathbf{A}$ *defined by recurrence (31).*

(a) $\mathbf{A}$ *is invertible.*

(b) $\mathbf{A}\mathbf{V}_\mathbf{M}^T$ *is invertible.*

(c) $a_i^{(n_i)}(\mathbf{M}[i,i])! = 0$.

(d) $\prod_{j\le i}(X - \mathbf{M}[j,j]) \not| a_i(X)$

We use Corollary 10.17 and in particular (d) to find conditions when $\mathbf{A}$ is invertible. For convenience, define $p_i(X) = \prod_{j<i}\mathbf{M}[j,j]$.

Consider a fixed i. We will show that if $d_{i+1},\dots,d_{N-1}$ are fixed such that (d) is satisfied for all $a_{[i+1:N]}(X)$, then we can choose d_i such that (d) is satisfied for $a_i(X)$ as well.

Case 1: There does not exist $j > i$ such that $\mathbf{M}[j,j] = \mathbf{M}[i,i]$.

By the recurrence,
$$a_i(X) = \frac{\sum_{j>i}\mathbf{M}[i,j]a_j(X)}{X - \mathbf{M}[i,i]} + d_i\frac{p_N(X)}{X - \mathbf{M}[i,i]}$$

Note that the first term on the RHS is a polynomial and also is a multiple of $p_i(X)$ by inductively invoking Lemma 10.14. Note that the second term is a multiple of $p_i(X)$ but not $p_{i+1}(X)$ by the assumption for this case. Thus there exists some d_i such that $\prod_{j\le i}(X - \mathbf{M}[j,j]) \not| a_i(X)$ holds - in fact, there is only one d_i such that it doesn't hold.

Case 2: There exists $j > i$ such that $\mathbf{M}[j,j] = \mathbf{M}[i,i]$. Choose j to be the largest such index.

Claim 1. Fix a_j and follow the recurrence (1) without adding multiples of the modulus - i.e. let $d_{[i:j]} = 0$. Let this family of polynomials be $a'_i(X)$. Then $p_{i+1}(X)|a_i(X)$ if and only if $p_{i+1}(X)|a'_i(X)$.

Proof. $a_i(X)$ and $a_i'(X)$ differ only through the terms $c_i p_N(X), \cdots, c_{j-1} p_N(X)$ added while following (1) with the modulus. All of these terms contribute a multiple of $p_{i+1}(X)$ to row i so $a_i(X) \equiv a_i'(X) \pmod{p_{i+1}(X)}$. □

Claim 2. $p_{i+1}(X) | a_i'(X)$ if and only if $p_{j+1}(X) | a_j(X)$.

Proof. The recurrence without the mod can be written as the product of $\Delta \times \Delta$ transition matrices, and in particular we can express $a_i'(X)$ via the equation

$$\frac{a_i'(X)}{p_i(X)} = h_{i,j}(X)\frac{a_j}{p_j} + h_{i,j+1}(X - \mathbf{M}[j,j])\frac{a_{j+1}}{p_{j+1}} + \cdots + h_{i,j+\Delta}(X - \mathbf{M}[j,j]) \cdots (X - \mathbf{M}[j+\Delta-1, j+\Delta-1])\frac{a_{j+\Delta}}{p_{j+\Delta}}$$

Multiplying through again to isolate a_i' and taking this mod $p_{i+1}(X)$, we see that only the first term affects whether $p_{i+1}(X) | a_i'(X)$.

Thus the claim is equivalent to saying that $h_{i,j}(X)$ is not a multiple of $(X - \mathbf{M}[i,i])$. If it was, then note that $a_i'(X)$ will *always* be a multiple of $p_{i+1}(X)$ no matter what $a_j(X)$ is. Then $a_i(X)$ will be a multiple of $p_{i+1}(X)$ no matter what $a_{i+1}, \cdots, a_{N-1}$ are, and there is *no* family of polynomials satisfying (d) of Corollary 10.17. This is a contradiction since there is some $\mathbf{A}$ satisfying recurrence (31) that is invertible - since there is a change of basis matrix $\mathbf{A}$ such that $\mathbf{M} = \mathbf{AFA}^{-1}$, and by Corollary 10.13 it satisfies (31). □

The results above thus imply the following result.

Theorem 10.18. *Let* $\mathbf{A}$ *be a matrix satisfying (31).*

- $\mathbf{AV}_{\mathbf{M}}^T$ *is upper triangular for any sequences* d_i.
- *We can pick* $d_{N-1}, \ldots, d_0$ *in order, such that each* d_i *chosen to satisfy the local condition* $a_i^{(n_i)}(M[i,i])! = 0$. *Then the matrix* $\mathbf{AV}_{\mathbf{M}}^T$ *will be invertible. Furthermore, if there exists* $j > i$ *such that* $M[i,i] = M[j,j]$ *then* d_i *can be anything, in particular* 0.

10.5.3 Finding the Error Coefficients and Inverting $\mathbf{AV_M}^T$

The main goal of this section is to prove a structure result on $\mathbf{AV}_{\mathbf{M}}^T$, that will easily allow us to

- Given a recurrence with unspecified error coefficients, pick the d_i such that $\mathbf{AV}_{\mathbf{M}}^T$ is invertible.
- Given a fully specified recurrence such that $\mathbf{AV}_{\mathbf{M}}^T$ is invertible, multiply $(\mathbf{AV}_{\mathbf{M}}^T)^{-1}\mathbf{b}$ fast.

Suppose we have fixed error coefficients $d_{[0:N]}$ and consider matrix $\mathbf{A}$ corresponding to the polynomials generated by a recurrence of the form (31)

$$(X - \lambda_i)a_i(X) = \sum_{j=1}^{\min(\Delta, N-i)} c_{i,j} a_{i+j}(X) + d_i p_{\lambda_{[0:N]}}(X) \tag{32}$$

(We will use the shorthand notation $p_{\alpha,\beta,\ldots}(X) = (X-\alpha)(X-\beta)\cdots$.)
By triangularity, we can partition the matrix as follows

$$\mathbf{AV}_{\lambda_{[0:N]}}^T = \left[\begin{array}{c|c} \mathbf{T}_L & \mathbf{B} \\ \hline 0 & \mathbf{T}_R \end{array}\right].$$

The core result is that the two triangular sub-blocks have the same structure as itself.

Lemma 10.19. *Given an N-size recurrence (32) that defines a matrix $\mathbf{A}$, there exist $N/2$-size recurrences of the form (32) producing matrices $\mathbf{A}_L, \mathbf{A}_R$ such that*

$$\mathbf{T}_L = \mathbf{A}_L \mathbf{V}^T_{\lambda_{[0:N/2]}} \quad \textit{and} \quad \mathbf{T}_R = \mathbf{A}_R \mathbf{V}^T_{\lambda_{[N/2:N]}} \mathbf{E}$$

for a matrix E that is a direct sum of upper-triangular Toeplitz matrices and invertible.

Furthermore, the coefficients of these recurrences can be found with $O(\Delta^2 N \log^3 N)$ operations.

Proof. Define $c_L(X) = \prod_{i=0}^{N/2-1}(X-\lambda_i)$ and $c_R(X) = \prod_{i=N/2}^{N-1}(X-\lambda_i)$, corresponding to the left and right halves of the diagonal elements.

Structure of $\mathbf{T}_R$

$\mathbf{T}_R$ consists of the higher order (derivative) evaluations of $a_{[N/2:N]}$. However, we would like to express it instead as low order evaluations (namely, corresponding to $\mathbf{V}_{\lambda_{[N/2:N]}}$) of low-degree polynomials. Fix a root λ of $p_{\lambda_{[N/2:N]}}$ and suppose that $\mathbf{T}_R$ "contains" the evaluations $a_i^{(j)}(\lambda), \dots, a_i^{(k)}$ for $k \ge j \ge 0$. Equivalently, j is the multiplicity of λ in $\lambda_{[0:N/2]}$ and $k-j$ is the multiplicity in $\lambda_{[N/2:N]}$.

Let $q_i(X) = a_i(X)/c_L(X)$ and consider the Taylor expansions of $a_i(X), q_i(X), c_L(X)$ around λ

$$a_i(X) = \sum_{\ell=j}^{\infty} \frac{\alpha_i^{(\ell)}(\lambda)}{\ell!}(X-\lambda)^\ell \qquad q_i(X) = \sum_{\ell=0}^{\infty} \frac{q_i^{(\ell)}(\lambda)}{\ell!}(X-\lambda)^\ell \qquad c_L(X) = \sum_{\ell=j}^{\infty} \frac{c_L^{(\ell)}(\lambda)}{\ell!}(X-\lambda)^\ell$$

Since polynomial multiplication corresponds to a convolution of coefficients,

$$\begin{bmatrix} a_i^{(j)}(\lambda)/j! \\ \vdots \\ a_i^{(k)}(\lambda)/k! \end{bmatrix} = \begin{bmatrix} q_i^{(0)}(\lambda)/0! \\ \vdots \\ q_i^{(k-j)}(\lambda)/(k-j)! \end{bmatrix} * \begin{bmatrix} c_L^{(j)}(\lambda)/j! \\ \vdots \\ c_L^{(k)}(\lambda)/k! \end{bmatrix}$$

where $*$ denotes the convolution. Consider the matrix $\mathbf{A}_R$ where row i consists of the coefficients of $q_i = a_i/c_L$. If S is the indices of the subsequence of $\lambda_{[N/2:N]}$ corresponding to λ, then the above equation can be equivalently written

$$(\mathbf{T}_R)_{i,S} = (\mathbf{A}_R \mathbf{V}^T_{\lambda_{[N/2:N]}})_{i,S} \begin{bmatrix} c_L^{(j)}(\lambda)/j! & \cdots & c_L^{(k)}(\lambda)/k! \\ \vdots & \ddots & \vdots \\ 0 & \cdots & c_L^{(j)}(\lambda)/j! \end{bmatrix}$$

There is an analogous upper-triangular Toeplitz matrix for each λ, so define $\mathbf{E}$ to be the $N \times N$ matrix that is the appropriate direct sum of these Toeplitz matrices, such that $(\mathbf{T}_R)_i = (\mathbf{A}_R \mathbf{V}^T_{\lambda_{[N/2:N]}})_i \mathbf{E}$ holds. Note that the unique entries of $\mathbf{E}$ correspond to the evaluation of c_L at the second half of $\mathbf{V}_{\lambda_{[0:N]}}$ which can be computed in $O(N \log^2 N)$. Also, $\mathbf{E}$ is upper-triangular with non-zero diagonal. Since the above equation holds for all $i \in [N/2 : N]$, we can factor

$$\mathbf{T}_R = (\mathbf{A}_R \mathbf{V}^T_{\lambda_{[N/2:N]}})\mathbf{E}$$

Finally, the polynomials that $\mathbf{A}_R$ correspond to satisfy a recurrence

$$(X-\lambda_i)q_i(X) = \sum_{j=1}^{\min(\Delta, N-i)} c_{i,j} q_{i+j}(X) + d_i p_{\lambda_{[N/2:N]}}(X)$$

which directly follows from dividing (32) by $p_{\lambda_{[0:N/2]}}$.

Structure of $\mathbf{T}_L$

$\mathbf{T}_L$ corresponds to evaluations of $a_{[0:N/2]}(X)$ and their derivatives at $\lambda_{[0:N/2]}$. Note that we can subtract $c_L(X)$ from any polynomial without changing these evaluations. So if we define the polynomials $q_i(X) : i \in [0 : N/2]$ to be the unique polynomials of degree less than $N/2$ such that $q_i = a_i \pmod{c_L}$ and $\mathbf{A}_L$ to be the corresponding matrix of coefficients of q_i, then $\mathbf{T}_L = \mathbf{A}_L \mathbf{V}^T_{\lambda_{[0:N/2]}}$.

It remains to show that the q_i satisfy a recurrence of the form (32) and to specify its coefficients. Then $\mathbf{A}_L$ will be parameterized the same way as $\mathbf{A}$, completing the self-similarity result that $\mathbf{T}_L$ has the same structure as $\mathbf{A}\mathbf{V}^T_{\lambda_{[0:N/2]}}$ but of half the size.

Consider an $i \in [0:N/2]$. Note that (32) implies that

$$\begin{aligned}(X-\lambda_i)a_i(X) &= \sum c_{i,j}a_{i+j}(X) \pmod{p_{\lambda_{[0:N]}}(X)}\\ (X-\lambda_i)a_i(X) &= \sum c_{i,j}a_{i+j}(X) \pmod{p_{\lambda_{[0:N/2]}}(X)}\\ (X-\lambda_i)q_i(X) &= \sum c_{i,j}q_{i+j}(X) \pmod{p_{\lambda_{[0:N/2]}}(X)}\end{aligned} \tag{33}$$

so that we know there exists scalars d'_i such that

$$(X-\lambda_i)q_i(X) = \sum_{j=1}^{\min(\Delta,N-i)} c_{i,j}q_{i+j}(X) + d'_i p_{\lambda_{[0:N/2]}}(X) \tag{34}$$

and the goal is to find these scalars. Define $b_i(X) = (a_i - q_i)/p_{\lambda_{[0:N/2]}}$ which is a polynomial from the definition of q_i. Subtract the previous equation from (32) and divide out $p_{\lambda_{[0:N/2]}}$ to obtain

$$(X-\lambda_i)b_i(X) = \sum_{j=1}^{\min(\Delta,N-i)} c_{i,j}b_{i+j}(X) + (d_i p_{\lambda_{[N/2:N]}}(X) - d'_i)$$

Therefore d'_i is the unique element of $\mathbb{F}$ that makes the RHS of the above equation divisible by $(X-\lambda_i)$. This element is just $\sum c_{i,j}b_{i+j}(\lambda_i) + d_i p_{\lambda_{[N/2:N]}}(\lambda_i)$. Note that $p_{\lambda_{[N/2:N]}}$ can be evaluated at all $\lambda_{[0:N/2]}$ in time $O(N\log^2 N)$ time, so that term can be treated separately. Then define $d''_i := d'_i - d_i p_{\lambda_{[N/2:N]}}(\lambda_i)$ which is equivalently defined as the unique number making $\sum_{j=1}^{\min(\Delta,N-i)} c_{i,j}b_{i+j}(X) - d''_i$ divisible by $(X-\lambda_i)$. It suffices to find the d''_i.

Proposition 10.20. *Suppose $b_N,\dots,b_{N+\Delta}$ are polynomials of degree N and $\lambda_i, c_{i,j}$ are some fixed scalars. For $i = N-1,\dots,0$, define b_i to be unique polynomial such that*

$$(X-\lambda_i)b_i(X) = \sum_{j=1}^{\min(\Delta,N-i)} c_{i,j}b_{i+j}(X) - d''_i$$

holds for some d''_i (which is also unique). Then we can find all the d''_i in time $O(\Delta^2 N\log^3 N)$.

Proof. If $N = O(\Delta)$, we can explicit compute the b_i and d''_i naively in time at most $O(\Delta^3)$. Otherwise we will find the d''_i with a divide-and-conquer algorithm.

As previously noted, the value of d''_i depends only on the values of $b_{[i+1:i+\Delta]}$ evaluated at λ_i. Conversely, the polynomial b_i contributes to the result only through its evaluations at $\lambda_{[i-\Delta:i-1]}$. In particular, $b_{[N/2+\Delta:N+\Delta]}$ can be reduced $\pmod{\prod_{[N/2:N]}(X-\lambda_j)}$ without affecting the d''_i. So the remainders of the starting conditions $b_{[N:N+\Delta]}$ mod $\prod_{[N/2:N]}(X-\lambda_j)$ suffice to recursively compute $d''_{[N/2:N]}$. Using these, we can use the ranged transition matrix to compute $b_{[N/2:N/2+\Delta]}$. Reducing these $\pmod{\prod_{[0:N/2]}(X-\lambda_j)}$ and recursing gives $d''_{[0:N/2]}$. The base cases need $O(\Delta^3 \cdot N/\Delta)$ operations which is dominated by the main recursion, which has runtime $T(N) = 2T(N/2) + \Delta^2 N\log^2 N$ resolving to $O(\Delta^2 N\log^3 N)$ assuming that the ranged transition matrices for the jumps are pre-computed. □

This auxiliary algorithm gives us the following result.

Proposition 10.21. *Given the coefficients of the recurrence (32) and starting conditions $a_{[N/2:N/2+\Delta]}$, we can find the coefficients of recurrence (34) in $O(\Delta^2 N\log^3 N)$ operations.*

Proof. First find $b_{[N/2:N/2+\Delta]}$ which reduces a_i mod $p_{\lambda_{[0:N/2]}}$. Then run the above algorithm to find all $d''_{[0:N/2]}$. Finally, set $d'_i = d''_i + d_i p_{\lambda_{[N/2:N]}}(\lambda_i)$. The bottleneck is the auxilliary algorithm which requires $O(\Delta^2 N\log^3 N)$ time. □

Corollary 10.22. *Given the coefficients of recurrence (34), we can find coefficients of (32) in $O(\Delta^2 N \log^3 N)$ operations.*

Proof. Same as above, but the last step is reversed. Given d'_i, we compute d_i as $d_i = (d'_i - d''_i)/p_{\lambda_{[N/2:N]}}(\lambda_i)$. This is well-defined if $p_{\lambda_{[N/2:N]}}(\lambda_i)$ is non-zero. Otherwise, λ_i appears later in the sequence and we can just set $d_i = 0$ by Theorem 10.18. □

This completes the proof of the self-similarity structure of sub-blocks of $\mathbf{AV}^T_{\lambda_{[0:N/2]}}$. □

Using Lemma 10.19, we can pick coefficients d_i for (32) that generate an invertible $\mathbf{A}$, and also multiply a vector by $\mathbf{A}^{-1}$.

Corollary 10.23. *Given a size-N recurrence (32) with fixed recurrence coefficients and starting conditions, and unspecified error coefficients d_i, we can pick the d_i such that the resulting matrix $\mathbf{A}$ is invertible in time $O(\Delta^2 N \log^4 N)$.*

Proof. If $N = O(\Delta)$, then we can explicitly compute $a_{N-1}, \dots, a_0$ in order using the recurrence while picking $d_{N-1}, \dots, d_0$ appropriately; the results of Section 10.5.2 ensure that this is possible. Otherwise, we use Lemma 10.19 to recurse.

Since $\mathbf{A}_R$ satisfies a size-$N/2$ recurrence with known recurrence coefficients and starting conditions and unknown error coefficients, we can recursively find $d_{[N/2:N]}$ such that $\mathbf{A}_R$ and hence $\mathbf{T}_R$ is invertible.

For the other half, we can jump the recurrence using $d_{[N/2:N]}$ to find $a_{[N/2:N/2+\Delta]}$ (time $O(\Delta^2 N \log^2 N)$). Now $\mathbf{A}_L$ satisfies a size-$N/2$ recurrence with known recurrence coefficients and unknown error coefficients, and we can use [corollary inside the lemma] to find d_i such that $\mathbf{A}_L$ is invertible.

Thus we have picked d_i such that $\mathbf{T}_L, \mathbf{T}_R$ are invertible, and by triangularity so is $\mathbf{AV}^T_{\lambda_{[0:N/2]}}$.

Reducing to the subproblems is $O(\Delta^2 N \log^3 N)$ work by Lemma 10.19. □

Corollary 10.24. *Given a fully specified recurrence (32) such that $\mathbf{AV}^T_{\lambda_{[0:N/2]}}$ is invertible, we can compute $(\mathbf{AV}^T_{\lambda_{[0:N/2]}})^{-1}\mathbf{b}$ for any vector $\mathbf{b}$ in $O(\Delta^2 N \log^4 N)$ operations.*

Proof. Note that inversion of a triangular block matrix can be expressed using the Schur complement as

$$\begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{0} & \mathbf{C} \end{bmatrix}^{-1} = \begin{bmatrix} \mathbf{A}^{-1} & -\mathbf{A}^{-1}\mathbf{B}\mathbf{C}^{-1} \\ \mathbf{0} & \mathbf{D}^{-1} \end{bmatrix}$$

So by Lemma 10.19, the desired product can be written as

$$(\mathbf{AV}^T_{\lambda_{[0:N/2]}})^{-1}\mathbf{b} = \begin{bmatrix} (\mathbf{A}_L\mathbf{V}^T_{\lambda_{[0:N/2]}})^{-1} & -(\mathbf{A}_L\mathbf{V}^T_{\lambda_{[0:N/2]}})^{-1}\mathbf{B}(\mathbf{A}_R\mathbf{V}^T_{\lambda_{[N/2:N]}}\mathbf{E})^{-1} \\ \mathbf{0} & (\mathbf{A}_R\mathbf{V}^T_{\lambda_{[N/2:N]}}\mathbf{E})^{-1} \end{bmatrix} \mathbf{b} = \begin{bmatrix} (\mathbf{A}_L\mathbf{V}^T_{\lambda_{[0:N/2]}})^{-1}\left(\mathbf{b}_L - \mathbf{B}\mathbf{E}^{-1}(\mathbf{A}_R\mathbf{V}^T_{\lambda_{[N/2:N]}})^{-1}\mathbf{b}_R\right) \\ \mathbf{E}^{-1}(\mathbf{A}_R\mathbf{V}^T_{\lambda_{[N/2:N]}})^{-1}\mathbf{b}_R \end{bmatrix}$$

where $\mathbf{b}_L = \mathbf{b}_{[0:N/2]}, \mathbf{b}_R = \mathbf{b}_{[N/2:N]}$.

So $(\mathbf{AV}^T_{\lambda_{[0:N/2]}})^{-1}\mathbf{b}$ can be computed with three matrix-vector multiplications: by $(\mathbf{A}_R\mathbf{V}^T_{\lambda_{[N/2:N]}})^{-1}$, $\mathbf{E}^{-1}$, $\mathbf{B}$, and $(\mathbf{A}_L\mathbf{V}^T_{\lambda_{[0:N/2]}})^{-1}$, in order. Note that multiplying by $\mathbf{E}^{-1}$ is easy since it is a direct sum of upper-triangular Toeplitz matrices. $\mathbf{B}$ is a submatrix of $\mathbf{A}$ which we can multiply in $\Delta^2(N \log^3 N)$ operations. Finally, the first and last multiplications are identical subproblems of half the size.

□

Thus we have shown:

Theorem 10.25. *Let $\mathbf{M}$ be an upper-triangular, Δ-banded matrix whose minimal polynomial equals its characteristic polyomial. Then M is $(\Delta^{\omega_{\mathscr{R}}}, \Delta^2)$ Jordan efficient. More precisely, there exists a Jordan decomposition $\mathbf{M} = \mathbf{AJA}^{-1}$ such that with $O(\Delta^{\omega}_{\mathscr{R}} N \log^2 N + \Delta^2 N \log^4 N)$ pre-processing time, multiplication by $\mathbf{A}, \mathbf{A}^{-1}$ can be computed in $O(\Delta^2 N \log^4 N)$ operations.*

10.6 Bit Complexity

It turns out that it is fairly easy to analyze the bit-complexity of our algorithms. In particular, we note that all the basic operations in our algorithms reduce to operations on polynomials. In particular, consider a matrix with recurrence width of t over the set of integers $\mathbb{Z}$. (We note that in most of the problems of computing sequences the input is indeed over $\mathbb{Z}$.) Note that our results on the efficiency of our algorithms are stated in terms of the number of operations of $\mathbb{Z}$. When dealing with the bit complexity of our algorithms, we have to worry about the size of the integers. It can be verified that given the input recurrence $(\mathbf{G}, \mathbf{F})$, all the integers are of size $(\max(\|\mathbf{F}\|_\infty, \|\mathbf{G}\|_\infty)^{O(N)}$. Since two n-bits integers are can multiplied with $O(n \log n \log\log n)$-bit operations, this implies to obtain the bit complexity of our algorithms, we just need to multiply our bounds on the number of operations by $\tilde{O}(N \cdot \max(\|\mathbf{F}\|_\infty, \|\mathbf{G}\|_\infty))$.

In particular, the above implies that for computing the Bernoulli numbers can be computed with $\tilde{O}(N^2)$-bit operations. This is within $\tilde{O}(1)$ factors of the best known algorithm developed specifically for this problem in [25].

10.7 Preliminary Experimental Results

We coded our basic algorithm in C++ and compared with a naive brute force algorithm. We would like to stress that in this section, we only present preliminary experimental results with the goal of showing that in principle the constants in our algorithms' runtimes are reasonable (at least to the extent that preliminary implementations of our algorithms beat the naive brute force algorithm).

We compare 3 facets of the algorithms: the preprocessing step, computing $\mathbf{Ab}$, and computing $\mathbf{A}^T\mathbf{b}$. We consider matrices $\mathbf{A}$ whose rows are the coefficients of polynomials that follow our basic recurrence:

$$f_{i+1}(X) = \sum_{i=0}^{t} g_{i,j}(X) f_i(X)$$

where $\deg(g_{i,j} = j, \deg(f_i) = i$. We generate the coefficients of $f_i(X)$ for $i \le t$, the coefficients of $g_{i,j}(X)$ for $i > t$, and the elements of $\mathbf{b}$ pseudo-randomly in the range `[-1, 1]` using the C++ `rand()` function. The input to the algorithms are the $f_i(X)$ for $i \le t$, $g_{i,j}(X)$ for $i > t$, and $\mathbf{b}$.

The brute force's preprocessing step is to explicitly compute the $\binom{N+1}{2}$ polynomial coefficients of $f_i(X)$ for all i, thereby computing the non-zero element of $\mathbf{A}$. The vector multiplication is the straightforward $O(N^2)$ algorithm. Our approach's preprocessing step uses the naive cubic matrix multiplication algorithm. And it uses the open-source library FFTW to compute FFTs.

The experiments below were run on a 2-year old laptop with an i7-4500U processor (up to 3 GHz, 4 MB cache) and 8 GB of RAM. All of the numbers below express times in seconds.

	Preprocessing		$\mathbf{Ab}$		$\mathbf{A}^T\mathbf{b}$	
	Brute Force	Our Approach	Brute Force	Our Approach	Brute Force	Our Approach
N = 100, t = 1	0.02	**0.01**	**0.0004**	0.003	**0.0006**	0.003
N = 100, t = 4	**0.04**	0.07	**0.0005**	0.009	**0.0005**	0.009
N = 1000, t = 1	1.3	**0.13**	0.05	**0.04**	0.06	**0.04**
N = 1000, t = 4	2.8	**1.2**	**0.04**	0.15	**0.05**	0.15
N = 10000, t = 1	127	**1.7**	4.8	**0.4**	5.6	**0.5**
N = 10000, t = 4	322.8	**18.2**	4.2	**1.8**	5.3	**1.8**

At $N = 100$, the brute force clearly outclasses ours, but notably our preprocessing isn't much slower than what brute force requires. Our approach starts to perform better at $N = 1000$ where the multiplication algorithms are similar in runtime to the brute force multiplication algorithms and are significantly faster than the brute force preprocessing. And at $N = 10000$, our approach universally outperforms the brute force approach.

11 Succinct Representations and Multivariate Polynomials

The goal of this section is two fold. The first goal is to present matrices that have low recurrence width in our sense but were not captured by previous notions of widths of structured matrices. The second goal is to show that if one can substantially improve upon the efficiency of our algorithms with respect to sharper notions of input size will leads to improvements in the state-of-the-art algorithms for multipoint evaluation of multivariate polynomials. Out initial interests in these matrices arose from their connections of coding theory, which we will also highlight as we deal with the corresponding matrices.

11.1 Multipoint evaluation of multivariate polynomials

We consider the following problem.

Definition 11.1. Given an m-variate polynomial $f(X_1,\dots,X_m)$ such that each variable has degree at most $d-1$ and $N = d^m$ distinct points $\mathbf{x}(i) = (x(i)_1,\dots,x(i)_m)$ for $1 \le i \le N$, output the vector $(f(\mathbf{x}(i)))_{i=1}^N$.

The best runtime for an algorithm that solves the above problem (over an arbitrary field) takes time $O(d^{\omega_2(m-1)/2+1})$, where an $n\times n$ matrix can be multiplied with an $n\times n^2$ matrix with $O(n^{\omega_2})$ operations [31,36]. (For the sake of completeness, we state these algorithms in Appendix B.) We remark on three points. First in the multipoint evaluation problem we do not assume any structures on the N points: e.g. if the points form an m-dimensional grid, then the problem can be solved in $\tilde{O}(N)$ many operations using standard FFT techniques. Second, if we are fine with solving the problem over *finite* fields, then the breakthrough result of Kedlaya and Umans [31] solves this problem with $N^{1+o(1)}$ operations (but for arbitrary N evaluation points). In other words, the problem is not fully solved only if we do not have any structure in the evaluation points and we want our algorithms to work over arbitrary fields (or even $\mathbb{R}$ or $\mathbb{C}$). Finally, from a coding theory perspective, this problem (over finite fields) corresponds to encoding of arbitrary puncturings of Reed-Muller codes.

Next, we aim to show that if we can improve our algorithms in certain settings then it would imply a fast multipoint evaluation of multivariate polynomials. In particular, we consider the following two more succinct ways of representing the input. We start with the more succinct way of representing the input. For a given polynomial $f(X) \in \mathbb{F}[X]$, let $\|f\|_0$ denote the size of the support of f. Finally, consider a matrix $\mathbf{A}$ defined by a recurrence in (24). Define

$$\|\mathbf{A}\|_0 = \sum_{i=0}^{N-1}\sum_{j=0}^{t} \|g_{i,j}\|_0 + rN,$$

i.e. the size of sum of the sizes of supports of $g_{i,j}$'s plus the size of the rank r-representation of the error matrix in (24).

The second less succinct representation where we have an extra bound that $\|g_{i,j}\| \le D$ (for potentially $D < t$) for the recurrence in (24). Then note that the corresponding matrix $\mathbf{A}$ can be represented with size $\Theta(tDN + rN)$ elements. In this case, we will explore if one can improve upon the dependence on r in Theorem 7.1.

We would like to point out that in all of the above the way we argue that the error matrix $\mathbf{E}$ has rank at most r is by showing it has at most r non-zero columns. Thus, for our reductions rN is also an upper bound on $\|\mathbf{E}\|_0$, so there is no hope of getting improved results in terms of the sparsity of the error matrix instead of its rank without improving upon the state-of-the-art results in multipoint evaluation of multivariate polynomials.

11.1.1 Multipoint evaluation of bivariate polynomials

We begin with the bivariate case (i.e. $m = 2$) since that is enough to connect improvements over our results to improving the state-of-the-art results in multipoint evaluation of bivariate polynomials.

For notational simplicity we assume that the polynomial is $f(X,Y) = \sum_{i=0}^{d-1}\sum_{j=0}^{d-1} f_{i,j}X^iY^j$ and the evaluation points are $(x_1,y_1),\dots,(x_N,y_N)$. Now consider the $N\times N$ matrix

$$\mathbf{A}^{(2)} = \begin{pmatrix} 1 & x_1 & \cdots & x_1^{d-1} & y_1 & y_1x_1 & \cdots & y_1x_1^{d-1} & y_1^2 & y_1^2x_1 & \cdots & y_1^2x_1^{d-1} & \cdots & y_1^{d-1} & y_1^{d-1}x_1 & \cdots & y_1^{d-1}x_1^{d-1} \\ 1 & x_2 & \cdots & x_2^{d-1} & y_2 & y_2x_2 & \cdots & y_2x_2^{d-1} & y_2^2 & y_2^2x_2 & \cdots & y_2^2x_2^{d-1} & \cdots & y_2^{d-1} & y_2^{d-1}x_2 & \cdots & y_2^{d-1}x_2^{d-1} \\ & & & & & & \vdots & & & & & & \vdots & & & & \\ 1 & x_N & \cdots & x_N^{d-1} & y_N & y_Nx_N & \cdots & y_Nx_N^{d-1} & y_N^2 & y_N^2x_N & \cdots & y_N^2x_N^{d-1} & \cdots & y_N^{d-1} & y_N^{d-1}x_N & \cdots & y_N^{d-1}x_N^{d-1} \end{pmatrix}.$$

Note that to solve the multipoint evaluation problem we just need to solve $\mathbf{A}^{(2)}\cdot\mathbf{f}$, where $\mathbf{f}$ contains the coefficients of $f(X,Y)$. Let $\mathbf{D}_X$ and $\mathbf{D}_Y$ denote the diagonal matrices with $\mathbf{x} = (x_1,\dots,x_N)$ and $\mathbf{y} = (y_1,\dots,y_N)$ on their diagonals respectively. Finally, define $\mathbf{Z} = \mathbf{S}^T$. Now consider the matrix

$$\mathbf{B}^{(2)} = \mathbf{D}_X^{-1}\mathbf{A}^{(2)} - \mathbf{A}^{(2)}\mathbf{Z}.$$

It can be checked that $\mathbf{B}^{(2)}$ has rank at most d. Indeed note that

$$\mathbf{B}^{(2)} = \begin{pmatrix} \frac{1}{x_1} & 0 & \cdots & 0 & \frac{y_1}{x_1} - x_1^{d-1} & 0 & \cdots & 0 & y_1\left(\frac{y_1}{x_1} - x_1^{d-1}\right) & 0 & \cdots & 0 & \cdots & y_1^{d-2}\left(\frac{y_1}{x_1} - x_1^{d-1}\right) & 0 & \cdots & 0 \\ \frac{1}{x_2} & 0 & \cdots & 0 & \frac{y_2}{x_2} - x_2^{d-1} & 0 & \cdots & 0 & y_2\left(\frac{y_2}{x_2} - x_2^{d-1}\right) & 0 & \cdots & 0 & \cdots & y_2^{d-2}\left(\frac{y_2}{x_2} - x_2^{d-1}\right) & 0 & \cdots & 0 \\ & & & & & & \vdots & & & & & & \vdots & & & & \\ \frac{1}{x_N} & 0 & \cdots & 0 & \frac{y_N}{x_N} - x_N^{d-1} & 0 & \cdots & 0 & y_N\left(\frac{y_N}{x_N} - x_N^{d-1}\right) & 0 & \cdots & 0 & \cdots & y_N^{d-2}\left(\frac{y_N}{x_N} - x_N^{d-1}\right) & 0 & \cdots & 0 \end{pmatrix}.$$

The above was already noticed in [37]. The above is not quite enough to argue what we want so we make the following stronger observation. Consider

$$\mathbf{C}^{(2)} = \mathbf{D}_Y^{-1}\mathbf{B}^{(2)} - \mathbf{B}^{(2)}\mathbf{Z}^d = \mathbf{D}_Y^{-1}\mathbf{D}_X^{-1}\mathbf{A}^{(2)} - \mathbf{D}_Y^{-1}\mathbf{A}^{(2)}\mathbf{Z} - \mathbf{D}_X^{-1}\mathbf{A}^{(2)}\mathbf{Z}^d - \mathbf{A}^{(2)}\mathbf{Z}^{d+1}. \tag{35}$$

We now claim that the rank of $\mathbf{C}^{(2)}$ is at most two. Indeed, note that

$$\mathbf{C}^{(2)} = \begin{pmatrix} \frac{1}{x_1 y_1} & 0 & \cdots & 0 & \frac{-x_1^{d-1}}{y_1} & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 & \cdots & 0 & 0 & \cdots & 0 \\ \frac{1}{x_2 y_2} & 0 & \cdots & 0 & \frac{-x_2^{d-1}}{y_2} & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 & \cdots & 0 & 0 & \cdots & 0 \\ & & & & & & \vdots & & & & & & \vdots & & & & \\ \frac{1}{x_N y_N} & 0 & \cdots & 0 & \frac{-x_N^{d-1}}{y_N} & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 & \cdots & 0 & 0 & \cdots & 0 \end{pmatrix}.$$

Note that we have $t = 1$, $D = d + 1$ and $r = 2$. Theorem 7.1 implies that we can solve the above problem with $\tilde{O}(d^3)$ operations. The algorithm of [36] uses $\tilde{O}(d^{\omega_2/2+1})$ many operations. However, note that

$$\|\mathbf{A}^{(2)}\|_0 = \Theta(d^2).$$

Thus, we have the following result:

Theorem 11.2. *If one can solve* $\mathbf{Ab}$ *for any* $\mathbf{b}$ *with* $\tilde{O}\left((\|\mathbf{A}\|_0)^{\omega_2/4+1/2-\epsilon}\right)$ *operations, then one will have an multipoint evaluation of bivariate polynomials with* $\tilde{O}(d^{\omega_2/2+1-2\epsilon})$ *operations, which would improve upon the currently best-known algorithm for the latter.*

11.1.2 Multipoint evaluation of multivariate polynomials

We now consider the general multivariate polynomial case. Note that we can represent the multipoint evaluation of the m-variate polynomial $f(X_1,\ldots,X_m)$ as $\mathbf{A}^{(m)}\mathbf{f}$, where $\mathbf{f}$ is the vector of coefficients and $\mathbf{A}^{(m)}$ is presented as follows.

Each of the d^m columns are indexed by tuples $\mathbf{i} \in \mathbb{Z}_d^m$ and the columns are sorted in lexicographic increasing order of the indices. Note that this implies that the $\mathbf{i} = (i_1,\ldots,i_m) \in \mathbb{Z}_d^m$ column is represented by

$$\mathbf{A}^{(m)}[:,\mathbf{i}] = \begin{pmatrix} \prod_{j=1}^m x(1)_j^{i_j} \\ \prod_{j=1}^m x(2)_j^{i_j} \\ \vdots \\ \prod_{j=1}^m x(N)_j^{i_j} \end{pmatrix},$$

where the evaluation points are given by $\mathbf{x}(1),\ldots,\mathbf{x}(N)$.

For notational simplicity, we will assume that m is even. (The arguments below can be easily modified for odd m.) Define recursively for $0 \le j \le m/2$:

$$\mathbf{B}^{(j)} = \mathbf{D}_{X_{m-j}}^{-1}\mathbf{B}^{(j+1)} - \mathbf{B}^{(j+1)}\mathbf{Z}^{d^j}, \tag{36}$$

where $\mathbf{D}_{X_k}$ is the diagonal matrix with $(x(1)_k, \ldots, x(N)_k)$ on its diagonal. Finally, for the base case we have

$$\mathbf{B}^{(\frac{m}{2}+1)} = \mathbf{A}^{(m)}.$$

It can be verified (e.g. by induction) that the recurrence in (36) can be expanded out to

$$\mathbf{B}^{(0)} = \sum_{S \subseteq [m/2,m]} (-1)^{m/2+1-|S|} \left(\prod_{j\in S} \mathbf{D}_{X_j}^{-1}\right) \mathbf{A}^{(m)} \left(\prod_{j\in[m/2,m]\setminus S} \mathbf{Z}^{d^{m-j}}\right). \tag{37}$$

We will argue that

Lemma 11.3. $\mathbf{B}^{(0)}$ *has rank at most* $2^{m/2} \cdot d^{m/2-1}$.

Note that the above lemma implies that the recurrence in (37) is a $\mathbf{Z}$-dependent recurrence with $t = 1$, $D = \frac{d^{1+m/2}-1}{d-1}$ and $r = 2^{m/2}d^{m/2-1}$. Note that in this case we have $tDN + rN = \Theta((2d)^{3m/2-1})$. Thus, we have the following result:

Theorem 11.4. *If for an* $\mathbf{Z}$*-dependent recurrence we could improve the algorithm from Theorem 7.1 to run with* $\tilde{O}(\text{poly}(t)\cdot DN + rN)$ *operations for matrix vector multiplication, then we would be able to solve the general multipoint evaluation of multivariate polynomials in time* $\tilde{O}((2d)^{3m/2-1})$*, which would be a polynomial improvement over the current best algorithm (when* $d = \omega(1)$*), where currently we still have* $\omega_2 > 3$.

Note that the above shows that improving the dependence in r in Theorem 7.1 significantly (even to the extent of having some dependence on Dr) will improve upon the current best-known algorithms (unless $\omega_2 = 3$).

11.1.3 Proof of Lemma 11.3

We now prove Lemma 11.3. We will argue that all but at most $2^{m/2}d^{m/2-1}$ columns of $\mathbf{B}^{(0)}$ are $\mathbf{0}$, which would prove the result. Towards this end we will use the expression in (37).

For notational convenience for any index $\mathbf{i} \in \mathbb{Z}_d^m$, we will use $X^{\mathbf{i}}$ to denote the monomial $\prod_{j=1}^m X_j^{i_j}$. Then note that $\mathbf{A}^{(m)}[:,\mathbf{i}]$ is just the evaluation of the monomial $X^{\mathbf{i}}$ on the points $\mathbf{x}(1),\ldots,\mathbf{x}(N)$. Further, it can be checked (e.g. by induction) that that exists polynomials $P_{\mathbf{i}}(X_1,\ldots,X_m)$ such that $\mathbf{B}^{(0)}[:,\mathbf{i}]$ is the evaluation of $P_{\mathbf{i}}(X_1,\ldots,X_m)$ on the points $\mathbf{x}(1),\ldots,\mathbf{x}(N)$.

To simplify subsequent expressions, we introduce few more notation. For any $S \subseteq [m/2, m]$, define the matrix

$$\mathbf{M}_S = \left(\prod_{j\in S} \mathbf{D}_{X_j}^{-1}\right) \mathbf{A}^{(m)} \left(\prod_{j\in[m/2,m]\setminus S} \mathbf{Z}^{d^{m-j}}\right). \tag{38}$$

We can again argue by induction that for every $\mathbf{i} \in \mathbb{Z}_d^m$, we have that $\mathbf{M}_s[:,\mathbf{i}]$ is the evaluation of a polynomial $Q_S^{\mathbf{i}}(X_1,\ldots,X_m)$ on the points $\mathbf{x}(1),\ldots,\mathbf{x}(N)$. Note that this along with (37), implies that

$$P_{\mathbf{i}}(X_1,\ldots,X_m) = \sum_{S\subseteq[m/2,m]} (-1)^{m/2+1-|S|} Q_S^{\mathbf{i}}(X_1,\ldots,X_m). \tag{39}$$

Now we claim that

Claim 3. For every $\mathbf{i} \in \mathbb{Z}_d^m$ that has an index $m/2 \le j^* \le m$ such that $i_{j^*} \ge 2$ and $S \subseteq [m/2, m] \setminus \{j^*\}$, the following holds:

$$Q_S^{\mathbf{i}}(X_1,\ldots X_m) = Q_{S\cup\{j^*\}}^{\mathbf{i}}(X_1,\ldots,X_m).$$

We first argue why the claim above completes the proof. Fix any $\mathbf{i} \in \mathbb{Z}_d^m$ that has an index $m/2 \le j^* \le m$ such that $i_{j^*} \ge 2$. Indeed by pairing up all $S \subseteq [m/2, m] \setminus \{j^*\}$ with $S \cup \{j^*\}$, Claim 3 along with (39) implies that

$$P_{\mathbf{i}}(X_1, \ldots, X_m) \equiv 0.$$

Note that this implies that $\mathbf{B}^{(0)}[:, \mathbf{i}] = \mathbf{0}$ if there exists an index $m/2 \le j^* \le m$ such that $i_{j^*} \ge 2$. Note that there are at least $d^m - 2^{m/2} \cdot d^{m/2-1}$ such indices, which implies that at most $2^{m/2} \cdot d^{m/2-1}$ non-zero columns in $\mathbf{B}^{(0)}$, as desired.

Proof of Claim 3. For any subset $T \subseteq [m]$, recall that $\mathbf{e}_T$ is the characteristic vector of T in $\{0,1\}^m$. Then note that for any $S \subseteq [m/2, m]$, we have

$$Q_S^{\mathbf{i}}(X_1, \ldots, X_m) = X^{(\mathbf{i} \ominus \mathbf{e}_{[m/2,m]\setminus S}) - \mathbf{e}_S},$$

where $\ominus$ is subtraction over $\mathbb{Z}_d^m$ (and $-$ is the usual subtraction over $\mathbb{Z}^m$). Indeed the $\ominus \mathbf{e}_{[m/2,m]\setminus S}$ term corresponds to the matrix $\left(\prod_{j \in [m/2,m]\setminus S} \mathbf{Z}^{d^{m-j}}\right)$ and the $-\mathbf{e}_S$ term corresponds to the matrix $\left(\prod_{j \in S} \mathbf{D}_{X_j}^{-1}\right)$ in (38).

Fix a $\mathbf{i} \in \mathbb{Z}_d^m$ that has an index $m/2 \le j^* \le m$ such that $i_{j^*} \ge 2$ and $S \subseteq [m/2, m] \setminus \{j^*\}$. Define $S' = S \cup \{j^*\}$. Then we claim that

$$(\mathbf{i} \ominus \mathbf{e}_{[m/2,m]\setminus S}) - \mathbf{e}_S = (\mathbf{i} \ominus \mathbf{e}_{[m/2,m]\setminus S'}) - \mathbf{e}_{S'},$$

which is enough to prove the claim. To prove the above, we will argue that

$$(\mathbf{i} \ominus \mathbf{e}_{[m/2,m]\setminus S}) = (\mathbf{i} \ominus \mathbf{e}_{[m/2,m]\setminus S'}) - \mathbf{e}_{j^*}.$$

Note that the above is sufficient by definition of S'. Now note that $\mathbf{e}_{[m/2,m]\setminus S} = \mathbf{e}_{[m/2,m]\setminus S'} + \mathbf{e}_{j^*}$. In particular, this implies that it is sufficient to prove

$$(\mathbf{i} \ominus \mathbf{e}_{[m/2,m]\setminus S'}) \ominus \mathbf{e}_{j^*} = (\mathbf{i} \ominus \mathbf{e}_{[m/2,m]\setminus S'}) - \mathbf{e}_{j^*}. \tag{40}$$

By the assumption that $i_{j^*} \ge 2$, we have that

$$(\mathbf{i} \ominus \mathbf{e}_{[m/2,m]\setminus S'})_{j^*} \ge 1,$$

which implies that both $\ominus \mathbf{e}_{j^*}$ and $-\mathbf{e}_{j^*}$ have the same effect on $(\mathbf{i} \ominus \mathbf{e}_{[m/2,m]\setminus S'})$, which proves (40), as desired. □

11.2 Multipoint evaluation of multivariate polynomials and their derivatives

In this section, we present another example of a matrix with our general notion of recurrence that has been studied in coding theory. We would like to stress that currently this section does not yield any conditional "lower bounds" along the lines of Theorem 11.2 or 11.4.

We begin by setting up the notation for the derivative of a multivariate polynomial $f(X_1, \ldots, X_m)$. In particular, given an $\imath = (i_1, \ldots, i_m)$, we denote the $\imath$th derivative of f as follows:

$$f^{(\imath)}(X_1, \ldots, X_m) = \frac{\partial^{i_1}}{\partial X_1} \cdots \frac{\partial^{i_m}}{\partial X_m} f,$$

where $\frac{\partial^0}{\partial X} f = f$. If the underlying field $\mathbb{F}$ is a finite field, then the derivatives are defined as the *Hasse* derivatives.

We consider the following problem.

Definition 11.5. Given an m-variate polynomial $f(X_1, \ldots, X_m)$ such that each variable has degree at most $d-1$, an integer $0 \le r < d$ and $n = d^m$ distinct points $\mathbf{a}(i) = (a(i)_1, \ldots, a(i)_m)$ for $1 \le i \le N$, output the vector

$$\left(f^{(\imath)}(\mathbf{a}(j))\right)_{j \in [n], \imath \in \mathbb{Z}_r^m}.$$

The above correspond to (puncturing) of multivariate multiplicity codes, which have been studied recently in coding theory [32–34]. These codes have excellent local and list decoding properties. We note that in definition of derivative codes d and r are limits on the total degree and the total order of derivatives while in our case these are bounds for individual variables. However, these changes the problem size by only a factor that just depends on m (i.e. we have at most a factor $m!$ more rows and columns). Since we think of m as constant, we ignore this difference. For such codes in [32, 34] the order of the derivatives r is assumed to be a constant. However, the derivative code used in [33], r is non-constant. We would like to mention that these matrices turn up in list decoding of Reed-Solomon and related codes (this was also observed in [37]).[9] In particular, for the Reed-Solomon list decoder of Guruswami and Sudan [23], the algorithm needs $\mathbf{A}$ with $m = 2$ and r being a polynomial in n.

We note that the problem above is the same as $\mathbf{A}\cdot\mathbf{f}$, where $\mathbf{f}$ is the vector of coefficients of $f(X_1,\ldots,X_m) = \sum_{J\in\mathbb{Z}_d^m} f_J X^J$, where as before X^J is the monomial $\prod_{\ell=1}^m X_j^{j_\ell}$ and the matrix $\mathbf{A}$ is defined as follows:

$$A_{(k,\imath),J} = \prod_{\ell=1}^m \binom{j_\ell}{i_\ell} (a(k)_\ell)^{j_\ell - i_\ell},$$

for $k\in[n]$, $J = (j_1,\ldots,j_m)\in\mathbb{Z}_d^m$ and $\imath = (i_1,\ldots,i_m)\in\mathbb{Z}_r^m$. We note that the definition holds over all fields.

We use the convention that $\binom{b}{c} = 0$ if $b < c$.

The aim of the rest of the section is to show that the matrix $\mathbf{A}$ satisfies a recurrence that we can handle.

For notational simplicity, let us fix $k\in[n]$ and we drop the dependence on k from the indices. In particular, consider the (submatrix):

$$A_{\imath,J} = \prod_{\ell=1}^m \binom{j_\ell}{i_\ell} (a_\ell)^{j_\ell - i_\ell}.$$

Think of $\imath = (\imath^0,\imath^1)$, where $\imath^0 = (i_1,\ldots,i_{m'})$ and $\imath^1 = (i_{m'+1},\ldots,i_m)$ for some $1\le m' < m$ to be determined. Now fix an $\imath = (\imath^0,\imath^1)$ such that $\imath^1 \neq \mathbf{0}$ and define

$$S_\imath = \{\ell\in\{m'+1,\ldots,m\} | i_\ell \neq 0\}.$$

Consider the following sequence of relations:

$$\begin{aligned}
A_{\imath,J} &= \left(\prod_{\ell\in[m]\setminus S_\imath}\binom{j_\ell}{i_\ell}(a_\ell)^{j_\ell-i_\ell}\right)\cdot\left(\prod_{\ell\in S_\imath}\left(\binom{j_\ell-1}{i_\ell-1}+\binom{j_\ell-1}{i_\ell}\right)a_\ell^{j_\ell-i_\ell}\right) &(41)\\
&= \sum_{T\subseteq S_\imath}\prod_{\ell=1}^m\binom{j_\ell-\mathbb{1}_{\ell\in S_\imath}}{i_\ell-\mathbb{1}_{\ell\in T}}a_\ell^{j_\ell-i_\ell} &(42)\\
&= \sum_{T\subseteq S_\imath}\left(\prod_{\ell=1}^m a_\ell^{\mathbb{1}_{\ell\in S_\imath}-\mathbb{1}_{\ell\in T}}\right)\cdot\left(\prod_{\ell=1}^m\binom{j_\ell-\mathbb{1}_{\ell\in S_\imath}}{i_\ell-\mathbb{1}_{\ell\in T}}a_\ell^{j_\ell-\mathbb{1}_{\ell\in S_\imath}-(i_\ell-\mathbb{1}_{\ell\in T})}\right)\\
&= \sum_{T\subseteq S_\imath}\mathbf{a}^{\mathbf{e}_{S_\imath}-\mathbf{e}_T}\cdot A_{\imath-\mathbf{e}_T,J-\mathbf{e}_{S_\imath}}. &(43)
\end{aligned}$$

In the above (41) follows from the following equality for integers $b\ge 0$ and $c\ge 1$, $\binom{b}{c} = \binom{b-1}{c-1}+\binom{b-1}{c}$ while (42) follows from the notation that $\mathbb{1}_P$ is the indicator value for the predicate P.

Consider the matrix $\mathbf{E}$ defined as follows:

$$\mathbf{E}[(k,\imath),:] = \mathbf{A}[(k,\imath),:] - \sum_{T\subseteq S_\imath}\mathbf{a}^{\mathbf{e}_{S_\imath}-\mathbf{e}_T}\cdot\mathbf{A}[(k,\imath-\mathbf{e}_T),:]\cdot\mathbf{S}^{\sum_{\ell\in S_\imath} d^{m-\ell}}. \quad (44)$$

By (43), we have that for every $\imath = (\imath^0,\imath^1)$ such that $\imath^1\neq\mathbf{0}$

$$E[(k,\imath),J] = 0.$$

In other words, the only non-zero rows of $\mathbf{E}$ are rows $\mathbf{E}[(k,\imath),:]$ with $\imath^1 = \mathbf{0}$. Thus, we have argued that

[9] However in the list decoding applications $\mathbf{A}$ is not square and one is interested in obtaining a non-zero element $\mathbf{f}$ from its kernel: i.e. a non-zero $\mathbf{f}$ such that $\mathbf{Af} = \mathbf{0}$.

Lemma 11.6. **E** *as defined in* (44) *has rank at most* $nr^{m'}$.

Proof. This follows from the fact that there are $n \cdot r^{m'}$ many indices $(k, \imath)$ with $\imath^1 = \mathbf{0}$. □

This in turn implies the following:

Lemma 11.7. *The recurrence in* (44) *has* $t = \frac{r^{m-m'+1}-1}{r-1}$, $D = \frac{d^{m-m'+1}-1}{d-1}$ *and rank* $nr^{m'}$.

11.2.1 Instantiation of parameters

We now consider few instantiation of parameters to get a feel for Lemma 11.7.

We start with the case of $n = 1$: note that in this case we have $r = d$ (since we want $N = n \cdot r^m = d^m$). In this case the choice of m' that makes all the parameters roughly equal is $m' = m/2$. In this case we get that (11.7) is an $(2d^{m/2}, 2d^{m/2}, d^{m/2})$-recurrence (i.e. $t = 2d^{m/2}$, $D = 2d^{m/2}$ and rank $d^{m/2}$). However, this does not give anything algorithmically interesting since the input size for such a recurrence is already $\Omega(d^{m/2} \cdot d^{m/2} \cdot N + d^{m/2} \cdot N) = \Omega(N^2)$.

Recall that a (T, D, R) recurrence has an input size of $(TD + R)N$. Thus, to decrease the input size of the recurrence in (44), we need to pick m' such that $2(m - m') = m'$. This implies that we pick $m' = 2m/3$ and thus we have an $(2d^{m/3}, 2d^{m/3}, d^{2m/3})$ recurrence for an overall input size of $O(d^{5m/6}) = O(N^{5/3})$.

References

[1] `https://en.wikipedia.org/wiki/Approximation_theory#Chebyshev_approximation`.

[2] `https://en.wikipedia.org/wiki/Jacobi_polynomials#Differential_equation`.

[3] `https://en.wikipedia.org/wiki/Zernike_polynomials`.

[4] `http://wis.kuleuven.be/events/OPSFA/`.

[5] `http://www.chebfun.org`.

[6] NIST Handbook of Mathematical Functions. Cambridge University Press, New York, NY, 2010, ch. 24. Print companion to [16].

[7] Aho, A. V., Hopcroft, J. E., and Ullman, J. D. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, 1974.

[8] Apostol, T. M. *Introduction to analytic number theory.* Springer Science & Business Media, 2013.

[9] Bella, T., Eidelman, Y., Gohberg, I., and Olshevsky, V. Computations with quasiseparable polynomials and matrices. *Theoretical Computer Science 409*, 2 (2008), 158 – 179. Symbolic-Numerical Computations.

[10] Bini, D., and Pan, V. Y. *Polynomial and Matrix Computations (Vol. 1): Fundamental Algorithms.* Birkhauser Verlag, Basel, Switzerland, Switzerland, 1994.

[11] Brent, R. P., and Kung, H. T. Fast algorithms for manipulating formal power series. *J. ACM 25*, 4 (Oct. 1978), 581–595.

[12] Cantor, D. G., and Kaltofen, E. On fast multiplication of polynomials over arbitrary algebras. *Acta Informatica 28*, 7 (1991), 693–701.

[13] Cao, Z., and Cao, H. On fast division algorithm for polynomials using newton iteration. In *International Conference on Information Computing and Applications* (2012), Springer, pp. 175–180.

[14] Chihara, T. *An Introduction to Orthogonal Polynomials.* Dover Books on Mathematics. Dover Publications, 2011.

[15] COOLEY, J. W., AND TUKEY, J. W. An algorithm for the machine calculation of complex fourier series. *Math. Comput. 19* (1965), 297–301.

[16] NIST Digital Library of Mathematical Functions. http://dlmf.nist.gov/, Release 1.0.11 of 2016-06-08. Online companion to [6].

[17] DRISCOLL, J. R., HEALY, JR., D. M., AND ROCKMORE, D. N. Fast discrete polynomial transforms with applications to data analysis for distance transitive graphs. *SIAM J. Comput. 26*, 4 (Aug. 1997), 1066–1099.

[18] DUMMIT, D. S., AND FOOTE, R. M. *Abstract algebra*, vol. 3. Wiley Hoboken, 2004.

[19] FIDUCCIA, C. M. An efficient formula for linear recurrences. *SIAM J. Comput. 14*, 1 (1985), 106–112.

[20] GERASOULIS, A. A fast algorithm for the multiplication of generalized hilbert matrices with vectors. *Mathematics of Computation 50*, 181 (1988), 179–188.

[21] GIORGI, P. On polynomial multiplication in chebyshev basis. *IEEE Trans. Comput. 61*, 6 (June 2012), 780–789.

[22] GOHBERG, I., AND OLSHEVSKY, V. Complexity of multiplication with vectors for structured matrices. *Linear Algebra and its Applications 202* (1994), 163 – 192.

[23] GURUSWAMI, V., AND SUDAN, M. Improved decoding of Reed-Solomon and algebraic-geometry codes. *IEEE Transactions on Information Theory 45*, 6 (1999), 1757–1767.

[24] HARTMAN, P. A lemma in the theory of structural stability of differential equations. *Proceedings of the American Mathematical Society 11*, 4 (1960), 610–620.

[25] HARVEY, D. A multimodular algorithm for computing bernoulli numbers. *Math. Comput. 79*, 272 (2010), 2361–2370.

[26] HIGHAM, N. J. *Functions of matrices: theory and computation.* Siam, 2008.

[27] HORN, R. A., AND JOHNSON, C. R., Eds. *Matrix Analysis.* Cambridge University Press, New York, NY, USA, 1986.

[28] KAILATH, T., KUNG, S.-Y., AND MORF, M. Displacement ranks of matrices and linear equations. *Journal of Mathematical Analysis and Applications 68*, 2 (1979), 395 – 407.

[29] KAILATH, T., AND SAYED, A. H. Displacement structure: Theory and applications. *SIAM Review 37*, 3 (1995), 297–386.

[30] KALTOFEN, E., AND SAUNDERS, B. D. On wiedemann's method of solving sparse linear systems. In *International Symposium on Applied Algebra, Algebraic Algorithms, and Error-Correcting Codes* (1991), Springer Berlin Heidelberg, pp. 29–38.

[31] KEDLAYA, K. S., AND UMANS, C. Fast polynomial factorization and modular composition. *SIAM J. Comput. 40*, 6 (2011), 1767–1802.

[32] KOPPARTY, S. List-decoding multiplicity codes. *Theory of Computing 11* (2015), 149–182.

[33] KOPPARTY, S., MEIR, O., RON-ZEWI, N., AND SARAF, S. High rate locally-correctable and locally-testable codes with sub-polynomial query complexity. *CoRR abs/1504.05653* (2015).

[34] KOPPARTY, S., SARAF, S., AND YEKHANIN, S. High-rate codes with sublinear-time decoding. *J. ACM 61*, 5 (2014), 28:1–28:20.

[35] MILLER, K. S. On linear difference equations. *The American Mathematical Monthly 75*, 6 (1968), 630–632.

[36] NÜSKEN, M., AND ZIEGLER, M. Fast multipoint evaluation of bivariate polynomials. In *Algorithms - ESA 2004, 12th Annual European Symposium, Bergen, Norway, September 14-17, 2004, Proceedings* (2004), pp. 544–555.

[37] OLSHEVSKY, V., AND SHOKROLLAHI, M. A. A displacement approach to efficient decoding of algebraic-geometric codes. In *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing, May 1-4, 1999, Atlanta, Georgia, USA* (1999), pp. 235–244.

[38] OLSHEVSKY, V., AND SHOKROLLAHI, M. A. Matrix-vector product for confluent cauchy-like matrices with application to confluent rational interpolation. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, May 21-23, 2000, Portland, OR, USA* (2000), pp. 573–581.

[39] PAN, V. Y. *Structured Matrices and Polynomials: Unified Superfast Algorithms.* Springer-Verlag New York, Inc., New York, NY, USA, 2001.

[40] PAN, V. Y., AND TSIGARIDAS, E. P. Nearly optimal computations with structured matrices. In *Symbolic-Numeric Computation 2014, SNC '14, Shanghai, China, July 28-31, 2014* (2014), pp. 21–30.

[41] PUTZER, E. J. Avoiding the jordan canonical form in the discussion of linear systems with constant coefficients. *The American Mathematical Monthly 73*, 1 (1966), 2–7.

[42] VANDEBRIL, R., BAREL, M. V., GOLUB, G., AND MASTRONARDI, N. A bibliography on semiseparable matrices*. *CALCOLO 42*, 3, 249–270.

[43] YU, K.-H., ZHANG, C., BERRY, G. J., ALTMAN, R. B., RÉ, C., RUBIN, D. L., AND SNYDER, M. Predicting non-small cell lung cancer prognosis by fully automated microscopic pathology image features. *Nature Communications 7* (2016).

[44] ZEILBERGER, D. A holonomic systems approach to special functions identities. *Journal of Computational and Applied Mathematics 32*, 3 (1990), 321 – 368.

[45] ZHLOBICH, P., BELLA, T., EIDELMAN, Y., GOHBERG, I., AND OLSHEVSKY, V. *Classifications of Recurrence Relations via Subclasses of (H,m)-quasiseparable Matrices*, vol. 15 of *Lecture Notes in Electrical Engineering.* Springer-Verlag GmbH, 2011, p. 23.

A Further definitions of recurrence width in matrices

We provide one more instantiation of the $\otimes$ operator that arises in applications. In some of our applications (for example displacement rank in Section 10.1), a matrix $\mathbf{R} \in \mathbb{F}^{N \times N}$ is fixed and vectors $\mathbf{f}_i$ are defined through a recurrence such as

$$D_{i+1}(\mathbf{R})\mathbf{f}_{i+1} = \sum_{j=0}^{t} g_{i,j}(\mathbf{R})\mathbf{f}_{i-j}$$

where all $D_{i+1}(\mathbf{R})$ are invertible or equivalently $\gcd(D_i, c_{\mathbf{R}}) = 1$. This equation is well-defined by $\mathbf{f}_{i+1} = \sum (D_{i+1}(\mathbf{R})^{-1} g_{i,j}(\mathbf{R}))\mathbf{f}_{i-j}$, which can be rewritten as

$$\mathbf{f}_{i+1} = \sum_{j=0}^{t} (g_{i,j}/D_{i+1}) \otimes \mathbf{f}_{i-j}$$

under the following definition.

Definition A.1. Let $\mathbf{R} \in \mathbb{F}^{N \times N}$ and let $\mathscr{R}$ be the subring of $\mathbb{F}(X)$ consisting of fractions whose denominators are not a multiple of $c_{\mathbf{R}}(X)$. Given $a(X) = (b(X)/c(X)) \in \mathscr{R}$, the evaluation $a(\mathbf{R})$ is naturally defined as $b(\mathbf{R})c(\mathbf{R})^{-1} = c(\mathbf{R})^{-1}b(\mathbf{R})$. Now define $a(X) \otimes \mathbf{z} = a(\mathbf{R})\mathbf{z}$.

However, note the following: Let $a(X), b(X) \in \mathbb{F}[X]$ such that $(b, c_{\mathbf{R}}) = 1$, and let $c(X) = a(X)b^{-1}(X) \pmod{c_{\mathbf{R}}(X)}$. It is straightforward to show that $a(\mathbf{R})/b(\mathbf{R}) = c(\mathbf{R})$ because $c_{\mathbf{R}}(\mathbf{R}) = 0$. Therefore it is also true that

$$\mathbf{f}_{i+1} = \sum_{j=0}^{t} (g_{i,j}/D_{i+1}) \otimes \mathbf{f}_{i-j}$$

under Definition 3.4. Importantly, here $g_{i,j}/D_{i+1}$ is treated as an element of $\mathbb{F}[X]/(c_{\mathbf{R}}(X))$ (in particular, it can be represented as a polynomial) instead of $\mathbb{F}(X)$ (where it is represented as a fraction). Thus Definition A.1 is more natural than but weaker than Definition 3.4, and when it arises we will automatically assume that the recurrence is interpreted under 3.4.

This is how we are formally defining recurrences such as (6): As written it considers matrices in a polynomial modulus which does not make sense, but we use it to represent defining a polynomial recurrence with coefficients in $\mathbb{F}[X]/(c_{\mathbf{R}}(X))$ and then evaluating at $\mathbf{R}$.

B Algorithms for multipoint evaluation of multivariate polynomials

Here we recollect known algorithms for multipoint evaluation of multivariate polynomials. Recall the multipoint evaluation problem:

Definition B.1. Given an m-variate polynomial $f(X_1, \ldots, X_m)$ such that each variable has degree at most $d-1$ and $N = d^m$ distinct points $\mathbf{x}(i) = (x(i)_1, \ldots, x(i)_m)$ for $1 \le i \le N$, output the vector $\left(f(\mathbf{x}(i))\right)_{i=1}^{N}$.

We will use the following reduction from [31, 36]. Let $\alpha_1, \ldots, \alpha_N$ be distinct points. For $i \in [m]$, define the polynomial $g_i(X)$ of degree at most $N-1$ such that for every $j \in [N]$, we have

$$g_i(\alpha_j) = x(j)_i.$$

Then as shown in [31], the multipoint evaluation algorithm is equivalent to computing the following polynomial:

$$c(X) = f(g_1(X), \ldots, g_m(X)) \mod h(X),$$

where $h(X) = \prod_{j=1}^{N}(X - \alpha_i)$. In particular, we have $c(\alpha_i) = f(\mathbf{x}(i))$ for every $i \in [N]$. Thus, we aim to solve the following problem:

Definition B.2 (Modular Composition)**.** Given a polynomial $f(X_1, \ldots, X_m)$ with individual degree at most $d-1$ and $m+1$ polynomials $g_1(X), \ldots, g_m(X), h(X)$ all of degree at most $N-1$ (where $N \overset{\text{def}}{=} d^m$), compute the polynomial

$$f(g_1(X), \ldots, g_m(X)) \mod h(X).$$

As was noted in [36], if for the multipoint evaluation problem all the $x(j)_1$ for $j \in [N]$ are distinct, then we can take $\alpha_j = x(j)_1$ and in this case $\deg(g_1) = 1$ since we can assume that $g_1(X) = X$. We will see that this allows for slight improvement in the runtime. Also in what follows, we will assume that an $n \times n$ and $n \times n^2$ matrix can be multiplied with $O(n^{\omega_2})$ operations.

B.1 Algorithm for the general case

Consider Algorithm 3 (which is a straightforward generalization of the algorithm for $m = 1$ from [11]).

We will use $X^{\imath}$ for any $\imath = (i_1, \ldots, i_m)$ to denote the monomial $\prod_{\ell=1}^{m} X_\ell^{i_\ell}$. Let k be any integer that divides d and define

$$q = \frac{d}{k}.$$

With this notation, write down f as follows

$$f(X_1, \ldots, X_m) = \sum_{\jmath \in \mathbb{Z}_q^m} \left(\sum_{\imath \in \mathbb{Z}_k^m} f_{\jmath,\imath} \cdot X^{\imath} \right) \cdot X^{\jmath \cdot k}, \tag{45}$$

Algorithm 3 Algorithm for Modular Composition: general case

```
Input: $f(X_1,\dots,X_m)$ in the form of (45) and $g_1(X),\dots,g_m(X),h(X)$ of degree at most $N-1$ with $N=d^m$
Output:
        $f(g_1(X),\dots,g_m(X)) \mod h(X)$
1: Let $k$ be an integer that divides $d$                    ▷ We will use $k=\sqrt{d}$
2: $q \leftarrow \frac{d}{k}$
3: For every $\imath=(i_1,\dots,i_m)\in\mathbb{Z}_k^m$ do
4:     $g_\imath(X) \leftarrow \prod_{\ell=1}^m \left(g_\ell(X)\right)^{i_\ell} \mod h(X)$.
5: For every $\jmath=(j_1,\dots,j_m)\in\mathbb{Z}_q^m$ do
6:     $g^\jmath(X) \leftarrow \prod_{\ell=1}^m \left(g_\ell(X)\right)^{j_\ell\cdot k} \mod h(X)$.
7: For every $\jmath\in\mathbb{Z}_q^m$ do
8:     $a_\jmath(X) \leftarrow \sum_{\imath\in\mathbb{Z}_k^m} f_{\jmath,\imath}\cdot g_\imath(X)$
9: Return $\sum_{\jmath\in\mathbb{Z}_q^m} a_\jmath(X)\cdot g^\jmath(X) \mod h(X)$
```

where $f_{\jmath,\imath}$ are constants.

Algorithm 3 presents the algorithm to solving the modular composition problem. The correctness of the algorithm follows from definition. We now argue its runtime.

Note that for a fixed $\imath\in\mathbb{Z}_k^m$, the polynomial $g_\imath(X)$ can be computed in $\tilde{O}(mN)$ operations since it involves m exponentiations and $m-1$ product of polynomials of degree at most $N-1$ mod $h(X)$. Thus, Step 3 overall takes $\tilde{O}(m\cdot k^m\cdot N)$ many operations. By a similar argument Step 5 takes $\tilde{O}(m\cdot q^m\cdot N)$ operations. Step 9 needs q^m polynomial multiplication (mod $h(X)$) and q^m-1 polynomial multiplication where all polynomial are of degree at most $N-1$ and hence, this step takes $\tilde{O}(q^m\cdot N)$ operations. So all these steps overall take $\tilde{O}(m\cdot\max(k,q)^m\cdot d^m)$ many operations.

So the only step we need to analyze is Step 7. Towards this end note that for any $\jmath\in\mathbb{Z}_q^m$

$$\begin{aligned} a_\jmath(X) &= \sum_{\imath\in\mathbb{Z}_k^m} f_{\jmath,\imath}\cdot g_\imath(X)\\ &= \sum_{\imath\in\mathbb{Z}_k^m} f_{\jmath,\imath}\cdot\sum_{\ell=0}^{N-1} g_\imath[\ell]\cdot X^\ell\\ &= \sum_{\ell=0}^{N-1}\left(\sum_{\imath\in\mathbb{Z}_k^m} f_{\jmath,\imath}\cdot g_\imath[\ell]\right)X^\ell. \end{aligned}$$

Thus, if we think of the $q^m\times d^m$ matrix $\mathbf{A}$, where $\mathbf{A}[\jmath,:]$ has the coefficients of $a_\jmath(X)$, then we have

$$\mathbf{A}=\mathbf{F}\times\mathbf{G},$$

where $\mathbf{F}$ is an $q^m\times k^m$ matrix with $F_{\jmath,\imath}=f_{\jmath,\imath}$ and $\mathbf{G}$ is an $k^m\times d^m$ matrix with $G_{\imath,\ell}=g_\imath[\ell]$. Let $\omega(r,s,t)$ be defined so that one can multiply an $n^r\times n^s$ with an $n^s\times n^t$ matrix with $n^{\omega(r,s,t)}$ operations. If we set $k=d^\epsilon$ for some $0\le\epsilon\le 1$, we have that Algorithm 3 can be implemented with

$$\tilde{O}\left(m\cdot d^{m(\max(\epsilon,1-\epsilon)+1)}+d^{m\cdot\omega(1-\epsilon,\epsilon,1)}\right)$$

many operations. It turns out that the expression above is optimized at $\epsilon=\frac{1}{2}$, which leads to an overall (assuming m is a constant) $\tilde{O}\left(d^{\omega_2 m/2}\right)$ many operations. Thus, we have argued that

Theorem B.3. *The modular composition problem with parameters d and m can be solved with $\tilde{O}\left(d^{\omega_2 m/2}\right)$ many operations.*

B.1.1 A 'direct' algorithm for the multipoint evaluation case

We now note that one can convert Algorithm 3 into a "direct" algorithm for the multipoint evaluation problem. Algorithm 4 has the details.

Algorithm 4 Algorithm for Multipoint Evaluation

Input: $f(X_1,\ldots,X_m)$ in the form of (45) and evaluation points $\mathbf{a}(i)$ for $i \in [N]$
Output:

$$\left(f(\mathbf{a}(i))\right)_{i\in[N]}$$

```
1: For every $\imath = (i_1,\ldots,i_{m/2}) \in \mathbb{Z}_d^{m/2}$ do
2:     $\mathbf{g}_\imath \leftarrow \left(\prod_{\ell=1}^{m/2} (a(k)_\ell)^{i_\ell}\right)_{k\in[N]}$
3: For every $\jmath = (j_1,\ldots,j_m) \in \mathbb{Z}_d^{m/2}$ do
4:     $\mathbf{g}^\jmath \leftarrow \left(\prod_{\ell=m/2+1}^{m} (a(k)_\ell)^{i_\ell}\right)_{k\in[N]}$
5: For every $\jmath \in \mathbb{Z}_d^{m/2}$ do
6:     $\mathbf{b}_\jmath \leftarrow \left(\sum_{\imath\in\mathbb{Z}_d^{m/2}} f_{\jmath,\imath} \cdot \mathbf{g}_\imath(k)\right)_{k\in[N]}$
7: Return $\sum_{\jmath\in\mathbb{Z}_d^{m/2}} \langle \mathbf{b}_\jmath, \mathbf{g}^\jmath \rangle$
```

The correctness of the algorithm again follows from definition. It is easy to check that the computation of $\mathbf{g}_\imath, \mathbf{g}_\jmath$ and the output vectors can be accomplished with $\tilde{O}(d^{3m/2})$ many operations. Finally, the computation of the $\mathbf{b}_\jmath$ can be done with $\tilde{O}(d^{\omega_2 m/2})$ many operations using fast rectangular matrix multiplication.

B.2 Algorithm for the distinct first coordinate case

We now consider the case when all the $x(j)_1$ for $j \in [N]$ are distinct: i.e. we assume that $g_1(X) = X$. In this case we re-write f as follows:

$$f(X_1,\ldots,X_m) = \sum_{\jmath\in\mathbb{Z}_q^{m-1}} \left(\sum_{\imath\in\mathbb{Z}_k^{m-1}} f_{\jmath,\imath}(X_1)\cdot X_{-1}^{\imath} \right) \cdot X_{-1}^{\jmath\cdot k}, \tag{46}$$

where $X_{-1}^{\imath}$ denotes the monomial on the variables $X_2,\ldots,X_m$ and each $f_{\jmath,\imath}(X_1)$ is of degree at most $d-1$.

Algorithm 5 Algorithm for Modular Composition: distinct first coordinate case

Input: $f(X_1,\ldots,X_m)$ in the form of (46) and $g_2(X),\ldots,g_m(X), h(X)$ of degree at most $N-1$ with $N = d^m$
Output:

$$f(X, g_2(X),\ldots,g_m(X)) \mod h(X)$$

```
1: Let $k$ be an integer that divides $d$                      ▷ We will use $k = \sqrt{d}$
2: $q \leftarrow \frac{d}{k}$
3: For every $\imath = (i_2,\ldots,i_m) \in \mathbb{Z}_k^{m-1}$ do
4:     $g_\imath(X) \leftarrow \prod_{\ell=2}^{m} \left(g_\ell(X)\right)^{i_\ell} \mod h(X)$.
5: For every $\jmath = (j_2,\ldots,j_m) \in \mathbb{Z}_q^{m-1}$ do
6:     $g^\jmath(X) \leftarrow \prod_{\ell=2}^{m} \left(g_\ell(X)\right)^{j_\ell\cdot k} \mod h(X)$.
7: For every $\jmath \in \mathbb{Z}_q^{m-1}$ do
8:     $a_\jmath(X) \leftarrow \sum_{\imath\in\mathbb{Z}_k^{m-1}} f_{\jmath,\imath}(X)\cdot g_\imath(X) \mod h(X)$
9: Return $\sum_{\jmath\in\mathbb{Z}_q^{m-1}} a_\jmath(X)\cdot g^\jmath(X) \mod h(X)$
```

Algorithm 5 shows how to update Algorithm 3 to handle this special case. Again the correctness of this algorithm follows from the definitions.

We next quickly outline how the analysis of Algorithm 5 differs from that of Algorithm 3. First, the same argument we used earlier can be used to show that Steps 3, 5 and 9 can be accomplished with $\tilde{O}(m \cdot \max(k,q)^{m-1} \cdot d^m)$ many operations.

As before the runtime is dominated by the number of operations needed for Step 7. Towards this end note that

$$\begin{aligned} a_J(X) &= \sum_{\iota \in \mathbb{Z}_k^{m-1}} f_{J,\iota}(X) \cdot g_\iota(X) \\ &= \sum_{\iota \in \mathbb{Z}_k^{m-1}} f_{J,\iota}(X) \cdot \sum_{\ell=0}^{d^{m-1}-1} g_\iota[\ell](X) \cdot (X^d)^\ell \\ &= \sum_{\ell=0}^{d^{m-1}-1} \left(\sum_{\iota \in \mathbb{Z}_k^{m-1}} f_{J,\iota}(X) \cdot g_\iota[\ell](X) \right) (X^d)^\ell. \end{aligned}$$

Note that in the above we have decomposed g_ι as a polynomial in powers of X^d (instead of X for Algorithm 3). In particular, this implies that all of $f_{J,\iota}(X)$ and $g_\iota[\ell](X)$ are polynomials of degree at most $d-1$. If we think of $a_J(X)$ as polynomials in X^d (with coefficients being polynomials of degree at most $2d-2$), we can represent the above as

$$\mathbf{A} = \mathbf{F} \times \mathbf{G},$$

where the $q^{m-1} \times k^{m-1}$ matrix $\mathbf{F}$ is defined by $F_{J,\iota} = f_{J,\iota}(X)$ and the $k^{m-1} \times d^{m-1}$ matrix $\mathbf{G}$ is defined by $g_{\iota,\ell} = g_\iota[\ell](X)$. Again if we set $k = n^\epsilon$, then the run time of Algorithm 5 is given by (we use the fact that each multiplication and addition in $\mathbf{F} \times \mathbf{G}$ can be implemented with $\tilde{O}(d)$ operations):

$$\tilde{O}\left(m \cdot d^{(m-1)\max(\epsilon,1-\epsilon)+m} + d \cdot d^{(m-1)\cdot\omega(1-\epsilon,\epsilon,1)}\right)$$

many operations. It turns out that the expression above is optimized at $\epsilon = \frac{1}{2}$, which leads to an overall (assuming m is a constant) $\tilde{O}\left(d^{1+\omega_2(m-1)/2}\right)$ many operations. Thus, we have argued that

Theorem B.4. *The modular composition problem with parameters d and m for the case of $g_1(X) = X$ can be solved with $\tilde{O}\left(d^{1+\omega_2(m-1)/2}\right)$ many operations.*