Abstract

TURNER, WILLIAM JONATHAN. Black Box Linear Algebra with the LinBox Library. (Under the direction of Erich Kaltofen.)

Black box algorithms for exact linear algebra view a matrix as a linear operator on a vector space, gathering information about the matrix only though matrix-vector products and not by directly accessing the matrix elements. Wiedemann's approach to black box linear algebra uses the fact that the minimal polynomial of a matrix generates the Krylov sequences of the matrix and their projections. By preconditioning the matrix, this approach can be used to solve a linear system, find the determinant of the matrix, or to find the matrix's rank. This dissertation discusses preconditioners based on Beneš networks to localize the linear independence of a black box matrix and introduces a technique to use determinantal divisors to find preconditioners that ensure the cyclicity of nonzero eigenvalues. This technique, in turn, introduces a new determinant-preserving preconditioner for a dense integer matrix determinant algorithm based on the Wiedemann approach to black box linear algebra and relaxes a condition on the preconditioner for the Kaltofen-Saunders black box rank algorithm. The dissertation also investigates the

minimal generating matrix polynomial of Coppersmith's block Wiedemann algorithm, how to compute it using Beckermann and Labahn's Fast Power Hermite-Padé Solver, and a block algorithm for computing the rank of a black box matrix. Finally, it discusses the design of the LinBox library for symbolic linear algebra.

BLACK BOX LINEAR ALGEBRA WITH THE LINBOX LIBRARY

BY

WILLIAM J. TURNER

A DISSERTATION SUBMITTED TO THE GRADUATE FACULTY OF
NORTH CAROLINA STATE UNIVERSITY
IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

APPLIED MATHEMATICS

COMPUTATIONAL MATHEMATICS CONCENTRATION

RALEIGH, NORTH CAROLINA

MAY 2002

APPROVED BY:

ERICH KALTOFEN
CHAIR OF ADVISORY COMMITTEE

HOON HONG

CARL D. MEYER

B. DAVID SAUNDERS

RALPH C. SMITH

To my wife, Cindy,

for all her love, patience, encouragement, and support.

Biography

William J. Turner grew up in Ankeny, Iowa, where he attended Ankeny High School until leaving at the end of his junior year for Iowa State University in Ames, Iowa. Using college courses, he graduated from high school with his class in 1992. In May 1994, he was awarded a Bachelor of Science degree with honors and distinction for a double major in Mathematics and Physics. He continued at Iowa State University to earn a Master of Science degree in Applied Mathematics in May 1996 before moving to North Carolina. He received a Doctor of Philosophy degree in Applied Mathematics with a Computational Mathematics Concentration in August 2002 from North Carolina State University in Raleigh, North Carolina. He is currently a Byron K. Trippet Assistant Professor of Computer Science in the Department of Mathematics and Computer Science at Wabash College in Crawfordsville, Indiana.

Acknowledgments

This dissertation would not have been possible without the help and support of many people. First and foremost, I am grateful to my advisor, Erich Kaltofen, for being both a mentor and a friend. His guidance led me to this research that I have found so enjoyable. Through it all, he has both challenged and supported me.

I appreciate also the support given to me by the members of my Ph.D. committee: Hoon Hong, Carl Meyer, B. David Saunders, and Ralph Smith. Their many comments and suggestions, as well as their various research backgrounds, have helped strengthened this dissertation.

I owe much to the members of the LinBox project. I especially thank Jean-Guillaume Dumas, Wayne Eberly, Mark Giesbrecht, Bradford Hovinen, Austin Lobo, Дмитрий Морозов (Dmitriy Morozov), and Gilles Villard.

Thank you to my fellow graduate students, both past and present, in the Department of Mathematics and the Symbolic Solutions Group for their encouragement and help. In particular, I thank Todd and Kristy Coffey, Wen-shin Lee, Markus Hitz, John May, Jack Perry, and George Yuhasz for their friendship and camaraderie.

I gratefully acknowledge the financial assistance I have received from the Department of Mathematics at North Carolina State University and the funding provided by the National Science Foundation.

Finally, none of this would have been possible without the love, patience, encouragement, and support of my family during these many years. I especially thank my parents and my wife, Cindy.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The black box approach to exact linear algebra views a matrix as a linear operator on a vector space (Kaltofen and Trager, 1990, Section 1). Black box algorithms only gather information about the black box matrix through matrix-vector products, and they do not change the matrix itself. (See Figure 1.1 on the following page.) The black box representation of matrices has long been present in numerical linear algebra. Textbooks such as Golub and Van Loan (1989) include many examples of these iterative methods, including the Lanczos methods (Lanczos, 1950, 1952) and the methods of conjugate gradients (Hestenes and Stiefel, 1952).

In symbolic computation, we desire algebraic methods to compute an exact solution, which is in contrast to the numerical approximation supplied in numerical analysis. Thus, algorithms from one discipline may not be suitable to the other. For instance, symbolic algorithms may be unstable numerically. On the other hand, the LaMacchia and Odlyzko

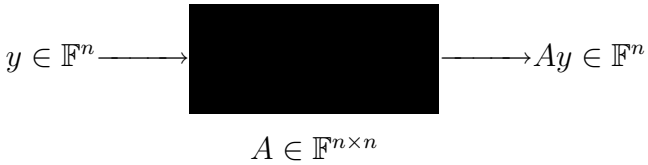

Figure 1.1: Black box matrix model

(1991) versions of the Lanczos and conjugate gradient methods come directly from numerical analysis, but self-orthogonal vectors can cause problems with Lanczos method over a finite field. Wiedemann (1986) introduces an algebraic method that finds relations in Krylov subspaces, and Lambert (1996) shows how the Wiedemann approach is related to the Lanczos method.

Black box linear algebra algorithms usually involve preconditioning the matrix to reduce a matrix problem to the computation of a minimal polynomial. This external view of a matrix is in contrast to most matrix algorithms, which take an internal view of the matrix and normally entail some elimination process. An elimination process, such as Gaussian elimination, changes the matrix while solving the problem, which can destroy useful structure and properties of the matrix. For example, Gaussian elimination may fill in a sparse matrix, causing the matrix to become dense. A black box algorithm will not do this because it does not change the matrix on which it acts.

Preconditioning of black box matrices is accomplished through a combination of pre- and post-multiplication by black box matrices,

$$\tilde{A} = B_1 A B_2.$$

Because of the black box matrix model, the preconditioned matrix $\tilde{A}$ will not be computed. Instead, the preconditioning matrices B_1 and B_2 will be used directly. To apply the preconditioned matrix $\tilde{A}$ to a vector x, we compute three matrix-vector products,

$$\tilde{A}x = B_1(A(B_2x)).$$

Thus, the preconditioning matrices B_1 and B_2 must have efficient matrix-vector products in addition to any other properties the problem may require.

Black box algorithms in symbolic computation must simultaneously meet three constraints on the resources they use. Algorithms must use no more than $O(n)$ black box matrix-vector products, $n^2(\log n)^{O(1)}$ additional arithmetic operations in $\mathbb{F}$, and $O(n)$ intermediate storage for field elements (Kaltofen, 2000, Section 3).

Many algorithms in symbolic computation make random choices during the computation. These randomized or probabilistic algorithms have some inherent uncertainty, but this uncertainty can be made boundedly small (von zur Gathen and Gerhard, 1999, Section 6.5). Two common classes of probabilistic algorithms are Monte Carlo and Las Vegas methods. Monte Carlo algorithms are always fast, but only probably correct. On the other hand, Las Vegas algorithms (Babai, 1979) are always correct, but only probably fast. A third class of algorithms, bounded probabilistic polynomial time (BPP) algorithms (Boppana and Hirschfeld, 1989), are probably correct and probably fast. BPP and Las Vegas methods can be converted to Monte Carlo methods by terminating the methods before completion. If a certificate exists to check the correctness of the answer

from a Monte Carlo algorithm, it can then be converted into a Las Vegas method by rerunning the algorithm until the solution passes the test. Unfortunately this scheme can create an infinite loop from a programming error, which cannot be distinguished from continually making bad random choices (Kaltofen, 2000, Section 3).

1.1 Wiedemann Method

In what is considered the seminal work in the field, Wiedemann (1986) introduces algorithms to solve a linear system and compute the determinant of a black box matrix over a finite field. Wiedemann's approach uses the fact that the minimal polynomial of a matrix generates the Krylov sequences of a matrix and their projections.

1.1.1 Nonsingular Linear Systems

The simplest problem in linear algebra is, perhaps, to solve the linear system

$$Ax = b \tag{1.1}$$

over a field $\mathbb{F}$ where the matrix $A \in \mathbb{F}^{n\times n}$ is square and nonsingular. In this case, the linear system (1.1) has a unique solution $x \in \mathbb{F}^n$ for each right-hand side vector $b \in \mathbb{F}^n$. Consider the Krylov sequence $\{A^i b\}_{i=0}^{\infty} \subset \mathbb{F}^n$ and its minimal polynomial

$$f^{A,b}(\lambda) = f^{A,b}[0] + f^{A,b}[1]\lambda + \cdots + f^{A,b}[m]\lambda^m \in \mathbb{F}[\lambda].$$

The vectors of the Krylov sequence satisfy the linear equation

$$f^{A,b}(A)b = f^{A,b}[0]b + f^{A,b}[1]Ab + \cdots + f^{A,b}[m]A^m b = 0. \tag{1.2}$$

Because the matrix A is nonsingular, the constant term of the minimal polynomial $f^{A,b}$ is nonzero, $f^{A,b}[0] \neq 0$, and the linear combination of Krylov vectors (1.2) can be rearranged to obtain the solution x to the nonsingular linear system (1.1)

$$x = A^{-1}b = -\frac{1}{f^{A,b}[0]}\left(f^{A,b}[1]b + f^{A,b}[2]Ab + \cdots + f^{A,b}[m]A^{m-1}b\right). \tag{1.3}$$

Wiedemann (1986, Section II) uses a vector $u \in \mathbb{F}^n$ to project the Krylov sequence to the bilinear projection sequence $\{u^{\mathrm{T}}A^i b\}_{i=0}^{\infty}$. Because the minimal polynomial $f^{A,b}$ of the Krylov sequence generates this sequence, the minimal polynomial $f_u^{A,b}$ of the bilinear projection sequence must divide the minimal polynomial $f^{A,b}$ of the Krylov sequence. If the vector u is chosen randomly from S^n where S is a finite subset of $\mathbb{F}$, these two minimal polynomials will be equal, $f_u^{A,b} = f^{A,b}$, with probability at least $1 - \deg(f^{A,b})/\left|S\right|$ (Kaltofen and Saunders, 1991, Lemma 1). Wiedemann computes the minimal polynomial $f_u^{A,b}$ of the bilinear projection using the Berlekamp-Massey algorithm (Berlekamp, 1968; Massey, 1969). One can easily check whether the minimal polynomial $f_u^{A,b}$ computed by the Berlekamp-Massey algorithm generates the Krylov sequence, in which case the two minimal polynomials are equal, $f_u^{A,b} = f^{A,b}$. Wiedemann gives two algorithms to correct for an unlucky random choice and compute $f^{A,b}$ from re-

peated applications of the Berlekamp-Massey algorithm. Once the minimal polynomial $f^{A,b}$ has been found, the solution to the linear system (1.1) can be computed from the linear combination of Krylov vectors (1.3).

1.1.2 Singular and Nonsquare Linear Systems

Wiedemann (1986, Section III) gives two methods to find the solution to the linear system (1.1) when the matrix A is singular or nonsquare. One method is to eliminate linearly dependent rows and columns of the matrix until an inconsistency or a square nonsingular system is found. This method requires using the minimal polynomial $f^{A,b}$ of the Krylov sequence to eliminate linearly dependent columns and applying the same method to the transposed matrix A^{T} to eliminate linearly dependent rows. A second method is to extend a nonsquare matrix A of full rank to a square nonsingular matrix by adjoining randomly selected rows or columns. Coppersmith (1994) gives another method to find the solution of the homogeneous linear system $Ax = 0$ when A is a square but singular matrix.

If the Jordan canonical form of the matrix A has no nilpotent blocks of size greater than one, the minimal polynomial of the matrix is divisible by λ, but not by λ^2. In other words,

$$f^A(\lambda) = f^A[1]\lambda + f^A[2]\lambda^2 + \cdots + f^A[m]\lambda^m \in \mathbb{F}[\lambda]$$

where $f^A[1] \neq 0$. If the linear system (1.1) is consistent, then, there exists a vector $y \in \mathbb{F}^n$

such that $b = Ay$, and

$$f^A[1]b + \cdots + f^A[m]A^{m-1}b = f^A[1]Ay + \cdots + f^A[m]A^m y = f^A(A)y = 0.$$

Thus,

$$\begin{aligned} b &= -\frac{1}{f^A[1]}\left(f^A[2]Ab + \cdots + f^A[m]A^{m-1}b\right) \\ &= -\frac{1}{f^A[1]}A\left(f^A[2]b + \cdots + f^A[m]A^{m-2}b\right) \end{aligned}$$

and a solution to the linear system is

$$x = -\frac{1}{f^A[1]}\left(f^A[2]b + \cdots + f^A[m]A^{m-2}b\right).$$

This means the singular nonhomogeneous linear system (1.1) can be solved by preconditioning the matrix A so that $\tilde{A}$ has no nilpotent blocks of size greater than one (Eberly and Kaltofen, 1997; Villard, 1998; Chen *et al.*, 2002).

Kaltofen and Saunders (1991, Section 4) give a method to uniformly sample the solution space of the square and singular linear system (1.1) if the rank of the matrix is known. They first precondition the matrix so that the leading $r \times r$ principal minor of $\tilde{A}$ is nonzero where $r = \text{rank}(A)$. Preconditioning the matrix to obtain a generic rank profile is sufficient. As Kaltofen and Saunders note in Section 4 of their paper, this can be done either with a parameterized matrix based on rearrangeable permutation networks

such as Beneš networks (Wiedemann, 1986, Section V),

$$\tilde{A} = A\,P, \tag{1.4}$$

where P is a parameterized matrix whose parameters can realize any column permutation matrix, or with unit upper and lower triangular Toeplitz matrices (Kaltofen and Saunders, 1991, Theorem 2),

$$\tilde{A} = T_1\,A\,T_2, \tag{1.5}$$

where T_1 is a unit upper triangular Toeplitz matrix,

$$T_1 = \begin{bmatrix} 1 & u_2 & \cdots & u_n \\ & \ddots & \ddots & \vdots \\ & & 1 & u_2 \\ & & & 1 \end{bmatrix},$$

and T_2 is a unit lower triangular Toeplitz matrix,

$$T_2 = \begin{bmatrix} 1 & & & \\ w_2 & 1 & & \\ \vdots & \ddots & \ddots & \\ w_n & \cdots & w_2 & 1 \end{bmatrix}.$$

Preconditioning transforms the original linear system (1.1) into a new linear system,

$$\tilde{A}\tilde{x} = \tilde{b}. \tag{1.6}$$

Kaltofen and Saunders choose a random column vector v and solve the $r \times r$ nonsingular linear system

$$\tilde{A}_r y'_{b,v} = \tilde{A}'(\tilde{b} + \tilde{A}v), \tag{1.7}$$

where $\tilde{A}_r$ is the leading principal $r \times r$ submatrix of $\tilde{A}$ and $\tilde{A}' \in \mathbb{F}^{r \times n}$ is the matrix formed by the first r rows of $\tilde{A}$. The solution to the $r \times r$ nonsingular linear system (1.7) gives the vector

$$y_{b,v} = \begin{bmatrix} y'_{b,v} \\ \\ 0 \end{bmatrix} \in \mathbb{F}^n$$

where $y_{b,v} - v$ uniformly samples the solution manifold of the preconditioned linear system (1.6).

In Chapter 2, we provide an improved generic rank profile preconditioner that combines the advantages of Wiedemann's parameterized matrix and Kaltofen and Saunders's Toeplitz matrices.

1.1.3 Black Box Matrix Rank

Wiedemann (1986, Section II) notes the algorithms for computing the minimal polynomial $f^{A,b}$ can be used to test probabilistically if A is singular. He expands this test

into a binary search method to find the rank of a singular matrix using randomly selected preconditioning matrices (Wiedemann, 1986, Section IV). Kaltofen and Saunders (1991, Section 4) improve on Wiedemann's approach to the rank problem by describing a new algorithm that is asymptotically faster than his binary search algorithm over large fields. They first precondition the matrix to place it into a generic rank profile. As in Section 1.1.2, a generic rank profile can be achieved by preconditioning either with a parameterized matrix based on rearrangeable permutation networks such as Beneš networks (Wiedemann, 1986, Section V), or with unit upper and lower triangular Toeplitz matrices (Kaltofen and Saunders, 1991, Theorem 2).

Once the matrix has a generic rank profile, Kaltofen and Saunders apply a diagonal multiplier so that, with high probability, the rank of the original singular matrix A is one less than the degree of the minimal polynomial of the preconditioned matrix $\tilde{A}$. Two matrix preconditioners for the Kaltofen-Saunders black box matrix rank algorithm are the preconditioner Wiedemann (1986, Section V) introduces for computing determinants,

$$\tilde{A} = A\,P \begin{bmatrix} d_1 & & \\ & \ddots & \\ & & d_n \end{bmatrix}, \tag{1.8}$$

where P is a parameterized matrix whose parameters can realize any column permutation

matrix, and Kaltofen and Saunders's rank preconditioner using Toeplitz matrices,

$$\tilde{A} = T_1 \, A \, T_2 \begin{bmatrix} d_1 & & \\ & \ddots & \\ & & d_n \end{bmatrix}, \tag{1.9}$$

where T_1 is a unit upper triangular Toeplitz matrix and T_2 is a unit lower triangular Toeplitz matrix. It is also possible to preserve Toeplitz-like structure by preconditioning the matrix with three Toeplitz matrices

$$\tilde{A} = T_3 \, A \, T_4 \, T_5, \tag{1.10}$$

where T_3 and T_4 are unit lower triangular Toeplitz matrices and T_5 is an upper triangular Toeplitz matrix (Chen *et al.*, 2002, Section 5).

The Kaltofen-Saunders rank algorithm is a Monte Carlo algorithm. There is currently no certificate for the rank of a black box matrix over an arbitrary field, although Saunders *et al.* (2001) introduce a certificate over a field of characteristic zero.

The improved generic rank profile preconditioner we present in Chapter 2 is applicable for the black box matrix rank problem in addition to the singular black box matrix system solver described in Section 1.1.2. In Chapter 3, we relax slightly the condition of placing the matrix into a generic rank profile before applying the diagonal matrix in the Kaltofen-Saunders rank algorithm.

1.1.4 Black Box Matrix Determinant

Wiedemann (1986, Section V) introduces an algorithm to compute the determinant of a black box matrix over an abstract field $\mathbb{F}$. For a random vector $v \in \mathbb{F}^n$, the minimal polynomial $f^{A,v}$ of the Krylov sequence $\{A^i v\}_{i=0}^{\infty}$ is equal to the minimal polynomial f^A of the matrix A with high probability. As we saw in Section 1.1.1, for a second random vector u, the minimal polynomial $f^{A,v}$ of the Krylov sequence is equal to the minimal polynomial $f_u^{A,v}$ of the bilinear projection sequence $\{u^{\mathrm{T}} A^i v\}_{i=0}^{\infty}$ with high probability. If the characteristic polynomial $\det(\lambda I - A)$ of the the matrix is equal to its minimal polynomial f^A, we have the relationship

$$\det(\lambda I - A) = f^A = f^{A,v} = f_u^{A,v}$$

with probability at least $1 - 2\deg(f^A)/|S|$ (Kaltofen and Pan, 1991, Lemma 2). This relation gives us a probabilistic algorithm for computing the determinant of the matrix A. We choose two random vectors u and v and use the Berlekamp-Massey algorithm to find the minimal polynomial $f_u^{A,v}$ of the bilinear projection sequence $\{u^{\mathrm{T}} A^i v\}_{i=0}^{\infty}$. The determinant is read from the constant term of this polynomial,

$$\det(A) = (-1)^n f_u^{A,v}(0).$$

Wiedemann's determinant algorithm requires the minimal polynomial of the matrix to equal its characteristic polynomial. In other words, the matrix must be cyclic. To

achieve this cyclicity, Wiedemann perturbs ("preconditions") the matrix by mixing the columns of the matrix to obtain a generic rank profile and then applying a diagonal multiplier. Wiedemann introduces a preconditioner (1.8) that uses a parameterized matrix and a diagonal multiplier for this purpose. However, Kaltofen and Saunders's rank preconditioner (1.9) can also be used for the determinant problem, and Kaltofen and Pan (1992, Proposition 1) show two Toeplitz matrices may be used without the diagonal matrix,

$$\tilde{A} = T_1 \, A \, T_2, \tag{1.11}$$

where T_1 is a unit upper triangular Toeplitz matrix and T_2 is a lower triangular Toeplitz matrix.

In Chen *et al.* (2002, Section 4), we improve on these preconditioners by showing a diagonal matrix is sufficient,

$$\tilde{A} = A \begin{bmatrix} d_1 & & \\ & \ddots & \\ & & d_n \end{bmatrix}, \tag{1.12}$$

and the generic rank profile is not required.

1.1.5 Dense Integer Matrix Determinant

The baby steps/giant steps algorithm of Kaltofen and Villard (2001, Section 2) provides a way to compute the determinant of a dense integer matrix using Wiedemann's determi-

nant algorithm and Chinese remaindering. The algorithm computes the determinant of the preconditioned matrix $\tilde{A}$ modulo several prime numbers using residue arithmetic and the Wiedemann determinant algorithm described in Section 1.1.4. It then calculates the determinant of the preconditioned matrix $\tilde{A}$ by the Chinese remainder algorithm. The algorithm is able to use the Wiedemann determinant algorithm efficiently by computing the bilinear projection sequence $\{u^{\mathrm{T}}A^i v\}_{i=0}^{\infty}$ using a "baby steps/giant steps" approach introduced by Kaltofen (1992b, Section 3) that computes $r = \lceil\sqrt{2n}\rceil$ baby steps $v_j = A^j v$ and $s = \lceil 2n/r\rceil$ giant steps $u_k = ((A^r)^{\mathrm{T}})^k u$.

Kaltofen and Villard's baby steps/giant steps algorithm is a Las Vegas algorithm. It bounds the determinant of the preconditioned matrix $\tilde{A}$ using Hadamard's inequality (Horn and Johnson, 1985, Section 7.8), and this bound determines how many moduli are required in the determinant's computation. Early termination could be used to create a Monte Carlo algorithm. Kaltofen (2002) gives an improved version of this Monte Carlo algorithm that dynamically changes the ratio of baby steps to giant steps. A preconditioner that increases the magnitude of the determinant requires both the Las Vegas and the Monte Carlo algorithms to use more moduli. Unfortunately, all of the preconditioners presented for the Wiedemann determinant algorithm in Section 1.1.4 increase the magnitude of the determinant. It is possible to bound the algorithm using the determinant of the original matrix A and avoid this increase in the number of moduli, but doing so imposes restrictions on the moduli that can be used by the algorithm. As Kaltofen and Villard (2001, Section 2) note, unit triangular Toeplitz matrices similar to

those in Kaltofen and Pan (1992, Section 2) will preserve the determinant. However, Toeplitz matrices are more costly to apply and consume more random field elements, which decreases the preconditioner's probability of success. Therefore, it is not clear that unit triangular Toeplitz matrices form a good preconditioner for these baby steps/ giant steps determinant algorithms.

In Chapter 3, we provide a new preconditioner that preserves the determinant. This preconditioner has the same probability of success as a diagonal matrix and can be applied far more cheaply than Toeplitz matrices. It also avoids any new restrictions on the moduli required in the determinant's computation.

1.2 Block Wiedemann Method

Coppersmith (1994) introduces a block version of Wiedemann's nonsingular system solver discussed in Section 1.1.1. Instead of using vectors $u \in \mathbb{F}^n$ and $v \in \mathbb{F}^n$, Coppersmith uses blocks of vectors $X \in \mathbb{F}^{n \times \beta_1}$ and $Y \in \mathbb{F}^{n \times \beta_2}$ to create a sequence of matrices $\{X^{\mathrm{T}} A^i Y\}_{i=0}^{\infty}$ instead of the scalar sequence $\{u^{\mathrm{T}} A^i v\}_{i=0}^{\infty}$ of Wiedemann's original algorithm. Coppersmith introduces a block Berlekamp-Massey algorithm to compute the right minimal matrix generating polynomial $F_X^{A,Y}$. This matrix polynomial can then be used to solve the singular homogeneous system $Ax = 0$ (Coppersmith, 1994) and to find the determinant of the matrix A (Kaltofen and Villard, 2001, Section 3).

Just as the Wiedemann and Lanczos methods are related (Lambert, 1996), the block Wiedemann method is related to the block Lanczos method discussed by Coppersmith

(1993) and Montgomery (1995).

In Chapter 4, we discuss linearly generated matrix sequences and the minimal generating polynomial, how to compute the minimal generating polynomial of a linearly generated matrix sequence using the Beckermann-Labahn Fast Power Hermite-Padé Solver (FPHPS) algorithm, and a block Monte Carlo algorithm for computing the rank of a matrix.

1.3 LinBox Library

Project LinBox is a collaboration of two dozen mathematicians and computer scientists in Canada, France, and the United States. The goal of the project is to produce algorithms and software for symbolic linear algebra, and black box linear algebra in particular. The project focuses primarily on the algorithms and not on implementing the underlying field arithmetic or the final interface a user might encounter.

The LinBox library is considered "middleware". (See Figure 1.2 on the next page, which is an adaptation of Figure 2.1 in Díaz (1997).) It is designed to be accessible to users outside the field of symbolic computation. For example, the library might be plugged into a commercial software package such as Mathematica or Maple using plug-and-play methodology (Kaltofen, 2000, Section 7). It must be designed in a way to allow these different platforms to access it. On the other hand, the library uses generic or reusable programming to access existing libraries to implement the field arithmetic needed in the library's algorithms, allowing the code to operate over many coefficient

domains and a variety of implementations for any given domain.

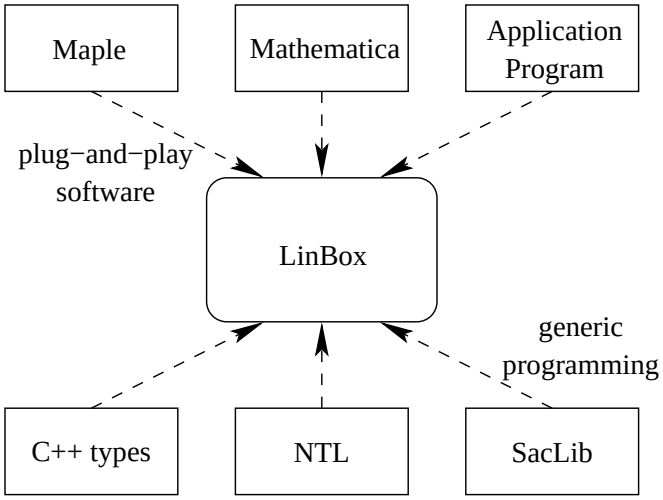

Figure 1.2: LinBox as middleware

The LinBox library follows, in part, the model of the Standard Template Library (STL) (Musser and Saini, 1996). The library primarily uses the C++ template mechanism (Stroustrup, 1997, Chapter 13) to implement the generic programming, with virtual functions (Stroustrup, 1997, Chapter 12) playing a secondary role. Much of the library's design is inspired by the FoxBox project (Díaz, 1997; Díaz and Kaltofen, 1998). The MTL [`http://www.osl.iu.edu/research/mtl`] numerical linear algebra project and the SYNAPS [`http://www-sop.inria.fr/galaad/logiels/synaps`] symbolic computation projects have similar goals. One predecessor of the LinBox library is the WiLiSys (Kaltofen and Lobo, 1999) implementation of the block Wiedemann algorithm (Dumas *et al.*, 2002).

In Chapter 5, we discuss the design of the LinBox library and the issues involved in

its creation. We presented much of this work in Dumas *et al.* (2002). Related work is also presented in Dumas (2000, Section 12.3).

Chapter 2

Preconditioners Based on Beneš Networks

The preconditioners for computing the rank of a black box matrix and randomly sampling the solution space of a singular black box system as discussed in Sections 1.1.3 and 1.1.2, respectively, need to precondition an $n \times n$ matrix of rank r so that the first r rows of the preconditioned matrix become linearly independent. In other words, the preconditioners must localize the linear independence of the system. Using the terminology of Chen *et al.* (2002, Section 3.1.1), the goal is to solve the PreCondInd problem. Two PreCondInd preconditioners can then be used to solve both the PreCondrxr and PreCondGen problems. These are the problems of ensuring the leading principal $r \times r$ minor of A is nonzero and ensuring all leading principal minors of A of size up to and including r are nonzero, respectively.

In his original paper, Wiedemann (1986, Section V) uses a parameterized matrix based on rearrangeable permutation networks such as Beneš networks to localize the linear independence of a black box matrix. A Beneš network is a network of butterfly switches that can realize any permutation of the inputs. A butterfly switch is a switch that takes two inputs and can either leave them as they are or exchange them. (See Figure 2.1.) Beneš (1964) shows the recursive nature of the network when n is a power of two.

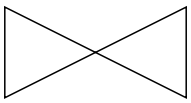

Figure 2.1: Butterfly switch

In this chapter (see also Chen *et al.*, 2002), we improve on the earlier work presented in Parker (1995) and Wiedemann (1986) in two ways. First, in Section 2.1, we use butterfly networks as in Parker (1995) instead of Beneš permutation networks as in Wiedemann (1986). A butterfly network is essentially half of a Beneš network. However, unlike Parker, we generalize our networks to an arbitrary size n and are not limited to powers of two. In Section 2.2, we show how to obtain preconditioners from these networks. Then, in Section 2.3, we generalize the networks again, this time by using switches of arbitrary radix.

2.1 Butterfly Networks

Let us consider the n rows of an $n \times n$ matrix of rank r. We want to precondition the matrix so the first r rows of the preconditioned matrix are linearly independent. We can use a switching network to exchange rows until the first r rows are linearly independent. Our goal is to switch any r rows of an arbitrary number n of rows to the beginning of the network. However, we must first consider the case of switching any r rows into any contiguous block when n is a power of two, $n = 2^l$.

Definition 2.1. An l-dimensional butterfly network is a recursive network of butterfly switches with 2^l nodes at each level such that at level m the node i is merged with node $i + 2^{m-1}$. Figure 2.2 illustrates a 3-dimensional butterfly with 8 nodes at each level. The bold lines demonstrate the switch settings needed to achieve the desired result.

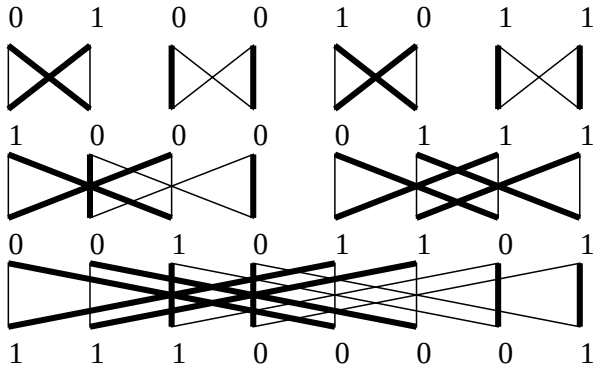

Figure 2.2: Butterfly network

We now show that when n is a power of two, a butterfly network can switch any r

rows into any contiguous block. This result will be used when we consider arbitrary n.

Lemma 2.1. *Let $n = 2^l$. The l-dimensional butterfly network of Definition 2.1 on the last page can switch any r indices $1 \leq i_1 < \cdots < i_r \leq n$ into any desired contiguous block of indices; wrapping the block around the outside, for our purposes, shall preserve contiguity. For example, in Figure 2.2 on the previous page, the ones in the final configuration would be considered contiguous. Furthermore, the network contains a total of $n \log_2(n)/2$ switches.*

Proof. Let us prove this lemma by induction. For $n = 1$, the proof is trivial because no switches are required.

Suppose the lemma is true for $n/2$. Let us divide the n nodes in half with r_1 given such that $i_{r_1} \leq n/2 < i_{r_1+1}$. We can construct butterfly networks of dimension $l - 1$ for each of these collections of $n/2$ nodes. By the lemma, each of these subnetworks can arrange the indices $i_1, \ldots, i_{r_1}$ and $i_{r_1+1}, \ldots, i_r$, respectively, into any desired contiguous block.

Let us consider the contiguous block desired from the network. It is either contained in the interior of the network in indices $1 \leq j, \ldots, j + r - 1 \leq n$, or it wraps around the outside of the network and can be denoted by indices $1, \ldots, j - 1$ and $n - r + j, \ldots, n$. This second situation can be converted into the first by instead thinking of switching the $n - r$ indices not originally chosen into the contiguous block $j, \ldots, j + n - r - 1$. Thus, we only have to consider the first situation. This can then be further divided into the two cases when the contiguous block $j, \ldots, j + r - 1$ is contained within one half and when

the block is in both halves and connected in the center.

For the first case, let us assume the desired block is completely within the first half, $j + r - 1 \leq n/2$. We can use the first subnetwork to place $i_1, \ldots, i_{r_1}$ so they switch into $j, \ldots, j + r_1 - 1$, and we can use the second subnetwork to position $i_{r_1+1}, \ldots, i_r$ to switch into $j + r_1, \ldots, j + r - 1$ as in Figure 2.3. A symmetric argument holds when the desired contiguous block is contained in the second half, $j > n/2$.

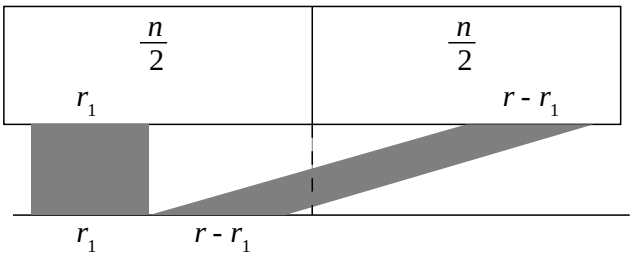

Figure 2.3: Butterfly network case 1

For the case when $j \leq n/2$ and $j + r - 1 > n/2$, let us assume $r_1 \leq n/2 - j + 1$, and, thus, we need to switch $r_2 = n/2 - j - r_1 + 1$ indices from the second half to the first. We can then use the first subnetwork to place $i_1, \ldots, i_{r_1}$ so they switch into $j, \ldots, j + r_1 - 1$, and we can use the second subnetwork to position $i_{r_1+1}, \ldots, i_r$ in a contiguous block that wraps around the outside of the subnetwork so they switch into $j + r_1, \ldots, j + r - 1$ as in Figure 2.4 on the next page. Once again, a symmetric argument holds for $r_1 \geq n/2 - j + 1$.

By the induction hypothesis, each of these subnetworks is of dimension $l - 1$ and has $n(\log_2(n) - 1)/4$ switches. Thus, the dimension of the final butterfly network is l, and

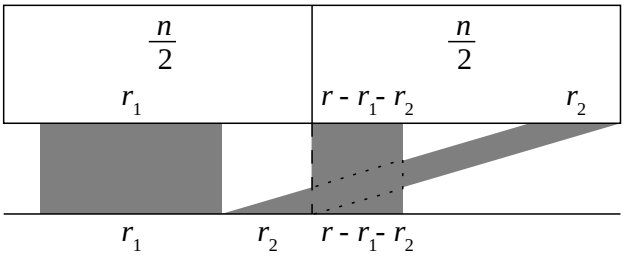

Figure 2.4: Butterfly network case 2

the total number of switches is

$$s = 2\frac{n}{4}\left(\log_2(n) - 1\right) + \frac{n}{2} = \frac{n}{2}\log_2(n). \qquad \blacksquare$$

Lemma 2.1 means we can switch any r rows of a $n \times n$ matrix into any contiguous block for $n = 2^l$. Now we are ready to consider our original goal of switching any r rows into the first block of r rows for any n. To do so, we must first describe a generalized butterfly network for any n.

Definition 2.2. When n is not a power of two, let us decompose n as a sum of powers of two,

$$n = \sum_{i=1}^{p} 2^{l_i} \text{ where } l_1 < l_2 < \cdots < l_p; \text{ let } n_i = 2^{l_i}. \tag{2.1}$$

First, we lay out butterfly networks for each of the n_i blocks. Then, we build a generalized butterfly network by connecting these butterfly networks using butterfly switches in a recursive manner such that $\sum_{i=1}^{p-1} n_i$ is merged with the far right nodes of n_p as in Figure 2.5 on the following page.

This generalized butterfly network can switch any r rows into the first block of r rows

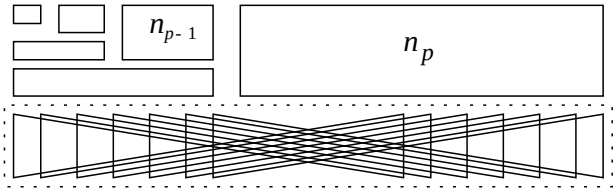

Figure 2.5: Generalized butterfly network

for any n.

Theorem 2.1. *The generalized butterfly network of Definition 2.2 on the previous page can switch any r indices $1 \leq i_1 < \cdots < i_r \leq n$ into the contiguous block $1, 2, \ldots, r$. Furthermore, it has a depth of $\lceil \log_2(n) \rceil$ and a total of no more than $n\lceil \log_2(n) \rceil/2$ butterfly switches. This bound is attained only when $n = 2^l$.*

Proof. If $n = 2^l$, the proof follows directly from Lemma 2.1 on page 22 and equality is obtained in the number of switches. Otherwise, we know $p > 1$ and $n_p > \sum_{i=1}^{p-1} n_i$ from the decomposition of n (2.1) . We prove the first part of this theorem by induction.

Suppose the theorem is true for $\sum_{i=1}^{p-1} n_i$. Let i_{r_1} be the last index in the left-most subnetwork, that is, $i_{r_1} \leq \sum_{i=1}^{p-1} n_i < i_{r_1+1}$. Then, we can switch the indices $i_1, \ldots, i_{r_1}$ into the contiguous block $1, \ldots, r_1$ using a generalized butterfly network.

If $r \leq \sum_{i=1}^{p-1} n_i$, we can use Lemma 2.1 on page 22 to position the remaining indices $i_{r_1+1}, \ldots, i_r$ so they switch into positions $r_1 + 1, \ldots, r$ as in Figure 2.6 on the following page. Otherwise let $r_2 = (\sum_{i=1}^{p-1} n_i) - r_1$, and then we can use the same lemma to position the indices as in Figure 2.7 on the next page.

The total number of butterfly switches is the number of switches for each of the subnetworks plus another $\sum_{i=1}^{p-1} n_i$ switches to combine the two. Another way of counting

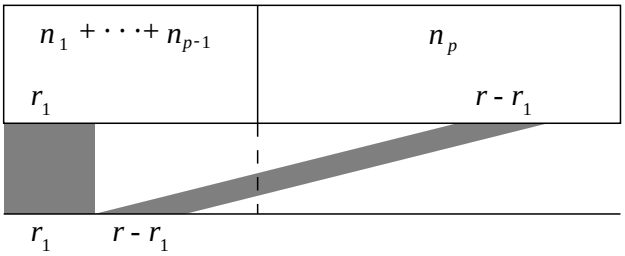

Figure 2.6: Generalized butterfly network case 1

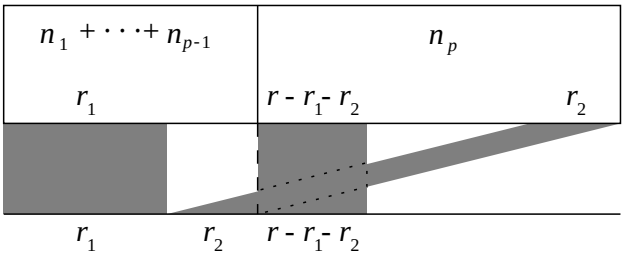

Figure 2.7: Generalized butterfly network case 2

the switches is the sum of the number of switches for each of the n_i blocks plus the number of switches to connect these blocks,

$$s = \sum_{i=1}^{p} \frac{n_i}{2} l_i + \sum_{i=1}^{p-1} \left(\sum_{j=1}^{i} n_j \right).$$

From the decomposition of n into powers of two (2.1), we know $l_i \leq l_p - (p - i)$ for $i < p$ and also $\sum_{j=1}^{i} n_j < n_{i+1}$. Thus,

$$s \leq \sum_{i=1}^{p} \frac{n_i}{2} l_p - \sum_{i=1}^{p} \frac{n_i}{2} (p - i) + \sum_{i=1}^{p-1} \left(\sum_{j=1}^{i} n_j \right),$$

and

$$s \leq \sum_{i=1}^{p} \frac{n_i}{2} l_p - \sum_{i=1}^{p-1} \left(\sum_{j=1}^{i} \frac{n_j}{2} \right) + \sum_{i=1}^{p-1} \left(\sum_{j=1}^{i} n_j \right) = \sum_{i=1}^{p} \frac{n_i}{2} l_p + \sum_{i=1}^{p-1} \left(\sum_{j=1}^{i} \frac{n_j}{2} \right)$$

because $\sum_{i=1}^{p} \left(\sum_{j=1}^{i} n_j \right) = \sum_{i=1}^{p} n_i (p - i)$. Therefore, again from the decomposition of n into powers of two (2.1) when $p > 1$, the total number of butterfly switches is

$$s < \sum_{i=1}^{p} \frac{n_i}{2} l_p + \sum_{i=2}^{p} \frac{n_i}{2} < \frac{n}{2} l_p + \frac{n}{2} = \frac{n}{2} \left(l_p + 1 \right) = \frac{n}{2} \lceil \log_2(n) \rceil.$$

This bound is never attained when $p > 1$. Furthermore, the depth of the network is $l_p + 1 = \lceil \log_2(n) \rceil$. ■

Theorem 2.1 is important because it generalizes the butterfly network of Lemma 2.1 on page 22 and Parker (1995) to an arbitrary size n. Previously, when $n \neq 2^l$, the matrix was extended to the new dimension $n' = 2^{\lceil \log_2(n) \rceil}$ before the butterfly network was applied. Theorem 2.1 shows a generalized butterfly network can be applied using fewer switches than would be required by a larger matrix. However, this decrease in the number of switches required comes at a cost. The generalized butterfly network is not able to switch the r inputs into any contiguous block when n is not a power of two.

2.2 Butterfly Network Preconditioners

Butterfly switching networks cannot be used directly to solve the PRECONDIND problem. Lemma 2.1 on page 22 and Theorem 2.1 on page 25 show there is a way to set the switches to achieve the desired result. The setting of the switches depends on knowing where the linearly independent rows of the matrix are found, which is information we do not have.

Wiedemann (1986, Section V) converts a permutation matrix into a parameterized matrix by replacing the butterfly switches with exchange matrices of the form

$$\begin{bmatrix} 1 & a \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ b & 1 \end{bmatrix} \begin{bmatrix} 1 & c \\ 0 & 1 \end{bmatrix}. \tag{2.2}$$

Instead of truly switching the rows of the matrix, this exchange matrix mixes the rows. By selecting random values for the matrix parameters, we are able to show the first r rows of the preconditioned matrix $\tilde{A} = BA$ are linearly independent with a certain probability if the original matrix has at least r linearly independent rows. Lobo (1995, Section 1.4.1) reduces this exchange matrix to a single variable,

$$\begin{bmatrix} 1-a & a \\ a & 1-a \end{bmatrix}. \tag{2.3}$$

Parker (1995) introduces the butterfly network described in Section 2.1 and an exchange matrix of the form

$$\begin{bmatrix} a & b \\ a & -b \end{bmatrix}. \tag{2.4}$$

Saunders (2001) uses a similar exchange matrix, but with the second row reversed, in his paper on least squares problems,

$$\begin{bmatrix} a & b \\ -b & a \end{bmatrix}. \tag{2.5}$$

In Chen *et al.* (2002, Section 6.2), we prove the generic exchange matrix

$$\hat{\mathcal{E}}(\alpha) = \begin{bmatrix} 1 & \alpha \\ 1 & 1+\alpha \end{bmatrix} \text{ with action } \hat{\mathcal{E}}(a) \begin{bmatrix} y^{[1]} \\ y^{[2]} \end{bmatrix} = \begin{bmatrix} y^{[1]} + ay^{[2]} \\ y^{[2]} + (y^{[1]} + ay^{[2]}) \end{bmatrix}$$

preserves the linear independence of the rows and can be used with the generalized butterfly networks of Theorem 2.1 on page 25 to create a PreCondInd preconditioner. This particular switch is important because it requires two additions and one multiplication, one less multiplication than the exchange matrices of both Parker (2.4) and Saunders (2.5). It also requires only one random variable and has determinant one.

2.3 Arbitrary Radix Switching Networks

In Section 2.1, we generalized the butterfly networks to arbitrary size n, but we are still using butterfly switches, which act on only two inputs. This means that if a switching

network composed of butterfly switches is to operate on more than two inputs, the network must be composed of more than one switch. In this section, we consider networks based on switches of arbitrary radix to reduce the total number of switches required. A radix-β switch is similar to a butterfly switch. It allows any permutation of the β inputs. Figure 2.8 shows switches or radix three and four.

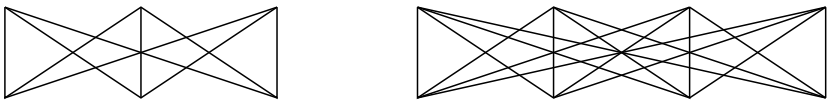

Figure 2.8: Radix-3 (left) and radix-4 (right) switches

As in Section 2.1, our goal is to use a radix-β switching network to switch any r rows of an arbitrary number n of rows to the beginning of the network. However, we must first consider the case of switching any r rows into any contiguous block when n is a power of β, $n = \beta^l$.

Definition 2.3. An l-dimensional radix-β switching network is a recursive network of radix-β switches with β^l nodes at each level such that at level m the nodes $i, i + \beta^{m-1}, \ldots, i + (\beta - 1)\,\beta^{m-1}$ $(1 \leq i \leq n/\beta)$ are merged. This network has l levels of switches with β^{l-1} switches at each level for a total of $l\,\beta^{l-1}$ switches in the network. The structure of this network is similar to that of the butterfly network shown in Figure 2.2 on page 21.

We are now ready for the radix-β counterpart to Lemma 2.1 on page 22.

Lemma 2.2. *Let $n = \beta^l$ where $\beta \geq 2$. The l-dimensional radix-β switching network of Definition 2.3 on the previous page can switch any r indices $1 \leq i_1 < \cdots < i_r \leq n$ into any desired contiguous block of indices; for our purposes, wrapping the block around the outside shall preserve contiguity. Furthermore, the network contains a total of $n \log_\beta(n)/\beta$ switches.*

Proof. Following the example of Lemma 2.1 on page 22, let us prove the lemma by induction. For $n = 1$, the proof is trivial because no switches are required.

Suppose the lemma is true for n/β. Let us divide the n nodes into β equal subnetworks with $r_1, r_2, \ldots, r_{\beta-1}$ given such that $i_{r_k} \leq kn/\beta < i_{r_k+1}$ for every k. Let us also denote $r_0 = 0$ and $r_\beta = r$. We can construct radix-β switching networks of dimension $l-1$ for each of these collections of n/β nodes. By the lemma, for every $1 \leq k \leq \beta$, the k-th subnetwork can arrange the indices $i_{r_{k-1}+1}, \ldots, i_{r_k}$ into any desired contiguous block.

Let us consider the contiguous block desired from the network. It is either contained in the interior of the network in indices $1 \leq j, \ldots, j+r-1 \leq n$, or it wraps around the outside of the network and can be denoted by indices $1, \ldots, j-1$ and $n-r+j, \ldots, n$. This second situation can be converted into the first by thinking of switching the $n-r$ indices not originally chosen into the contiguous block $j, \ldots, j+n-r-1$. Thus, we only have to consider the first situation.

We can use the first subnetwork to place $i_1, \ldots, i_{r_1}$ so they switch into $j, \ldots, j+r_1-1$. We can then use the second subnetwork to place $i_{r_1+1}, \ldots, i_{r_2}$ so they switch into $j+r_1, \ldots, j+r_2-1$. We can continue in this manner until we use the β-th subnetwork to

place $i_{r_{\beta-1}+1}, \ldots, i_r$ so they switch into $j+r_{\beta-1}, \ldots, j+r-1$. In each of these subnetworks, the indices may wrap around the outside of the subnetwork to allow them to switch into the correct position as was needed in case 2 of the proof of Lemma 2.1 on page 22. (See Figure 2.4 on page 24.)

Using the induction hypothesis, each of these subnetworks is of dimension $l-1$ and has $n(\log_\beta(n) - 1)/\beta^2$ switches. Thus, the dimension of the final radix-β switching network is l, and the total number of switches required is

$$s = \beta\frac{n}{\beta^2}\left(\log_\beta(n) - 1\right) + \frac{n}{\beta} = \frac{n}{\beta}\log_\beta(n). \qquad \blacksquare$$

As with Lemma 2.1 on page 22 when $\beta = 2$, Lemma 2.2 means we can switch any r rows of an $n \times n$ matrix into any contiguous block when $n = \beta^l$. We must now consider a network of arbitrary size n.

Definition 2.4. When n is not a power of the radix β, let us decompose n as a sum of powers of β,

$$n = \sum_{i=1}^{p} c_i\,\beta^{l_i} \text{ where } c_i \in \{1, \ldots, \beta-1\} \text{ and } l_1 < l_2 < \cdots < l_p; \text{ let } n_i = c_i\,\beta^{l_i}. \qquad (2.6)$$

First, we lay out radix-β switching networks for each of the β^{l_i} blocks. Then, we build a generalized radix-β switching network by connecting these networks with switches recursively, such that $\sum_{i=1}^{p-1} n_i$ is merged with the far right nodes of each of the β^{l_p} blocks with switches of radix $c_p + 1 \le \beta$. If $c_p > 1$, the remaining $n_p - \sum_{i=1}^{p-1} n_i$ nodes of each

of the c_p blocks of size β^{l_p} are merged with switches of radix $c_p < \beta$. These two sets of extra switches do not interact, so they constitute only one extra layer of switches after the last level of radix-β switching networks.

Another way to think of this level of connecting switches is to think of it as one set of switches of radix $c_p + 1 \leq \beta$ that switch the corresponding outputs of the c_p radix-β networks with the combined outputs of the generalized radix-β network for $\sum_{i=1}^{p-1} n_i$ and $\beta^p - \sum_{i=1}^{p-1} n_i$ "phantom" outputs to the left of the generalized subnetwork. In this way, we see there is only one additional level to connect all of the subnetworks.

Before we consider a completely arbitrary network size n, let us first consider the case when $n = c\,\beta^l$ where $1 < c < \beta$. This is a case that does not arise when $\beta = 2$.

Lemma 2.3. *Let $n = c\,\beta^l$ where $\beta \geq 2$ and $1 \leq c \leq \beta - 1$. The generalized radix-β switching network of Definition 2.4 on the last page can switch any r indices $1 \leq i_1 < \cdots < i_r \leq n$ into any desired contiguous block of indices; for our purposes, wrapping the block around the outside shall preserve contiguity. Furthermore, it has a depth of $\lceil \log_\beta(n) \rceil$ and a total of no more than $\beta^{\lceil \log_\beta(n) \rceil} \lceil \log_\beta(n) \rceil / \beta$ switches of radix no greater than β. This bound is attained only when $c = 1$ or $l = 0$.*

Proof. If $c = 1$, the proof is directly from Lemma 2.2 on page 30. Otherwise, let $r_1, r_2, \ldots, r_{c-1}$ be given such that $i_{r_k} \leq kn/\beta < i_{r_k+1}$ for every k. Let us also denote $r_0 = 0$ and $r_c = r$. We can construct radix-β switching networks of dimension l for each of these collections of β^l nodes. By Lemma 2.2, for every $1 \leq k \leq c$, the k-th radix-β network can arrange the indices $i_{r_{k-1}+1}, \ldots, i_{r_k}$ to any desired contiguous block.

As in the proof of Lemma 2.2, let us consider the contiguous block desired from the network. It is either contained in the interior of the network in indices $1 \leq j, \ldots, j + r - 1 \leq n$ or it wraps around the outside of the network and can be denoted by indices $1, \ldots, j - 1$ and $n - r + j, \ldots, n$. This second situation can be converted into the first by instead switching the $n - r$ indices not originally chosen into the contiguous block $j, \ldots, j + n - r - 1$. Thus, we once again only have to consider the first situation.

We can use the first radix-β network to place $i_1, \ldots, i_{r_1}$ so they switch into $j, \ldots, j + r_1 - 1$. We can then use the second radix-β network to place $i_{r_1+1}, \ldots, i_{r_2}$ so they switch into $j + r_1, \ldots, j + r_2 - 1$. We can continue in this manner until we use the c-th radix-β network to place $i_{r_{c-1}+1}, \ldots, i_r$ so they switch into $j + r_{c-1}, \ldots, j + r - 1$. In each of these radix-β networks, the indices may wrap around the outside of the network to allow them to switch into the correct position as was needed in case 2 of the proof of Lemma 2.1 on page 22. (See Figure 2.4 on page 24.)

The number of switches needed when $c \neq 1$ is the number for the c radix-β networks plus β^l switches to combine the networks,

$$s = \frac{c\,\beta^l}{\beta} l + \beta^l \leq \frac{\beta\,\beta^l}{\beta} l + \beta^l = \frac{\beta^{l+1}}{\beta}\,(l+1) = \frac{\beta^{\lceil \log_\beta(n) \rceil}}{\beta} \lceil \log_\beta(n) \rceil.$$

This bound is never attained when $c \neq 1$ and $l \neq 0$. Furthermore, the depth of the network for $c \neq 1$ is $l + 1 = \lceil \log_\beta(n) \rceil$. ■

The network uses fewer than $\beta^{\lceil \log_\beta(n) \rceil} \lceil \log_\beta(n) \rceil / \beta$ switches unless $n = \beta^l$ or $n = c$.

We are now ready to present the radix-β counterpart to Theorem 2.1 on page 25 for

an arbitrary network size n.

Theorem 2.2. *Let $\beta \geq 2$. The generalized radix-β switching network discussed in Definition 2.4 on page 32 can switch any r indices $1 \leq i_1 < \cdots < i_r \leq n$ into the contiguous block $1, 2, \ldots, r$. Furthermore, it has a depth of $\lceil \log_\beta(n) \rceil$ and a total of no more than $\beta^{\lceil \log_\beta(n) \rceil} \lceil \log_\beta(n) \rceil / \beta$ switches of radix no greater than β. This bound is attained only when $n = \beta^{\lceil \log_\beta(n) \rceil}$ or $n < \beta$.*

Proof. The proof of this theorem is very similar to the proof of Theorem 2.1 on page 25.

If $n = c\,\beta^l$, the proof follows directly from Lemma 2.3 on page 33. Otherwise, we know $p > 1$ and $n_p > \sum_{i=1}^{p-1} n_i$ from the decomposition of n into powers of β (2.6) . Following the example of Theorem 2.1 on page 25, we prove the first part of the theorem by induction. Suppose the theorem is true for $\sum_{i=1}^{p-1} n_i$. Let i_{r_1} be the last index in the left-most subnetwork, $i_{r_1} \leq \sum_{i=1}^{r-1} n_i < i_{r_1+1}$. Then, we can switch the indices $i_1, \ldots, i_{r_1}$ into the contiguous block $1, \ldots, r_1$ using a generalized butterfly network.

Let $r_2, \ldots, r_{c-1}$ be given such that, for every k, i_{r_k} is the last of the chosen indices before the end of the first $(k-1)$-st radix-β network,

$$i_{r_k} \leq (k-1)\,\beta^l + \sum_{i=1}^{r-1} n_i < i_{r_k+1}.$$

Let us also denote $r_\beta = r$. As in the proofs of Theorem 2.1 and Lemma 2.3, we can use the first radix-β network to place $i_1, \ldots, i_{r_1}$ so they switch into $j, \ldots, j + r_1 - 1$. Then, we can use the second radix-β network to place $i_{r_1+1}, \ldots, i_{r_2}$ so they switch into

$j + r_1, \ldots, j + r_2 - 1$. We can continue in this manner until we use the c-th radix-β network to place $i_{r_{c-1}+1}, \ldots, i_r$ so they switch into $j + r_{c-1}, \ldots, j + r - 1$. In each of these radix-β networks, the indices may wrap around the outside of the network to allow them to switch into the correct position as was needed in case 2 of the proof of Theorem 2.1 on page 25 (See Figure 2.7 on page 26.)

The total number of butterfly switches is the number of switches for each of the subnetworks plus the number of switches to combine them. When $c_p = 1$, this is an additional $\sum_{i=1}^{p-1} n_i < \beta^{l_p}$ switches. When $c_p \neq 1$, the number of additional switches needed is β^l. Another way of counting the switches is the sum of the number of switches for each of the n_i blocks plus the number of switches to connect these blocks,

$$s = \sum_{i=1}^{p} \frac{c_i \, \beta^{l_i} l_i}{\beta} + \sum_{i=1}^{p-1} \left(\sum_{j=1}^{i} c_j \, \beta^{l_j} \right) + \sum_{\substack{i=2 \\ c_i > 1}}^{p} \left(\beta^{l_i} - \sum_{j=1}^{i-1} c_j \, \beta^{l_j} \right).$$

In other words,

$$s \leq \sum_{i=1}^{p} \frac{c_i \, \beta^{l_i} l_i}{\beta} + \sum_{i=2}^{p} \beta^{l_i} = \sum_{i=1}^{p} \frac{n_i}{\beta} l_i + \sum_{i=2}^{p} \beta^{l_i}.$$

From the decomposition of n into powers of β (2.6), we know $l_i \geq i - 1$ for $1 \leq i \leq p$, so

$$\sum_{i=2}^{p} \beta^{l_i} \leq \sum_{i=2}^{l_p+1} \beta^{i-1} = \sum_{i=1}^{l_p} \beta^i$$

and

$$\sum_{i=1}^{p} \frac{n_i}{\beta} l_i \leq \sum_{i=1}^{p} \frac{(\beta - 1)\beta^{l_i}}{\beta} l_i \leq \sum_{i=0}^{l_p} \frac{(\beta - 1)\beta^i}{\beta} i = \sum_{i=1}^{l_p} \frac{(\beta - 1)\beta^i}{\beta} i.$$

Thus,

$$\sum_{i=1}^{p} \frac{n_i}{\beta} l_i \leq \sum_{i=1}^{l_p} \frac{(\beta-1)\beta^i}{\beta} l_p - \sum_{i=1}^{l_p-1} \frac{(\beta-1)\beta^i}{\beta}(l_p - i).$$

However, $\sum_{i=0}^{l_p-1}(\beta-1)\beta^i = \beta^{l_p} - 1$, so

$$\sum_{i=1}^{l_p} \frac{(\beta-1)\beta^i}{\beta} l_p = \sum_{i=0}^{l_p-1}(\beta-1)\beta^i l_p = \beta^{l_p} l_p - l_p$$

and

$$\begin{aligned}\sum_{i=1}^{l_p-1} \frac{(\beta-1)\beta^i}{\beta}(l_p - i) &= \sum_{i=1}^{l_p-1}\left(\sum_{j=1}^{i} \frac{(\beta-1)\beta^j}{\beta}\right) = \sum_{i=1}^{l_p-1}\left(\sum_{j=1}^{i}(\beta-1)\beta^{j-1}\right) \\ &= \sum_{i=1}^{l_p-1}\left(\beta^i - 1\right) = \sum_{i=1}^{l_p-1}\beta^i - l_p + 1.\end{aligned}$$

Therefore, the number of switches required when $p > 1$ is given by

$$\begin{aligned}s &\leq \sum_{i=1}^{p} \frac{n_i}{\beta} l_i + \sum_{i=2}^{p} \beta^{l_i} \leq \beta^{l_p} l_p - \sum_{i=1}^{l_p-1} \beta^i - 1 + \sum_{i=1}^{l_p} \beta^i \\ &< \beta^{l_p} l_p + \beta^{l_p} = \frac{\beta^{l_p+1}(l_p+1)}{\beta} = \frac{\beta^{\lceil \log_\beta(n) \rceil} \lceil \log_\beta(n) \rceil}{\beta}.\end{aligned}$$

This bound is never attained if $p > 1$. However, Lemma 2.3 on page 33 tells us this bound is attained when $p = 1$ and $c = 1$ or $l = 0$. These are the cases when $n = \beta^{\lceil \log_\beta(n) \rceil}$ and $n < \beta$.

The depth of the network when $p > 1$ is the depth of the largest subnetworks of a power of three plus one level to combine the rest of the network with these subnetworks. Thus, the complete network has a depth of $l_p + 1 = \lceil \log_\beta(n) \rceil$. ■

2.4 Arbitrary Radix Network Preconditioners

As in Section 2.2, these arbitrary radix switching networks cannot be used directly to solve the PreCondInd problem. Once again, the switching of rows must be converted into a mixing that preserves the desired linear independence. Unfortunately, finding a good radix-β exchange matrix is not easy. For this discussion, let us consider the $\beta \times \beta$ matrix

$$\mathcal{E}(\alpha_1, \ldots, \alpha_{\beta^2}) = \begin{bmatrix} \alpha_1 & \alpha_2 & \cdots & \alpha_\beta \\ \alpha_{\beta+1} & \alpha_{\beta+2} & \cdots & \alpha_{2\beta} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_{(\beta-1)\beta+1} & \alpha_{(\beta-1)\beta+2} & \cdots & \alpha_{\beta^2} \end{bmatrix}, \tag{2.7}$$

where $\alpha_1, \ldots, \alpha_{\beta^2}$ are indeterminates over the field. It is possible to choose values $a_1, \ldots, a_{\beta^2}$ for $\alpha_1, \ldots, \alpha_{\beta^2}$ to achieve any desired permutation of the inputs. Thus, this symbolic matrix (2.7) can be used as a radix-β exchange matrix. For the k-th switch in the generalized radix-β network, we can embed this symbolic exchange matrix (2.7) into an $n \times n$ identity matrix as was done by Chen *et al.* (2002, Section 6) to obtain the matrix $\hat{\mathcal{E}}_k$. This may not be the best switching matrix, but we can use it to prove the following counterpart to Theorem 6.3 in Chen *et al.* (2002).

Theorem 2.3. *Let $\mathbb{F}$ be a field, let A be an $n \times n$ matrix over $\mathbb{F}$ with r linearly independent rows, let s be the number of switches in the generalized radix-β switching network from Theorem 2.2 on page 35, and let S be a finite subset of $\mathbb{F}$. Let N be the number of random numbers required in this network. If $a_1, \ldots, a_N$ are randomly chosen uniformly*

and independently from S then the first r rows of

$$\left(\prod_{k=1}^{s} \hat{\mathcal{E}}_k\right) A$$

are linearly independent with probability no less than

$$1 - \frac{r\lceil \log_\beta(n) \rceil}{|S|} \geq 1 - \frac{n\lceil \log_\beta(n) \rceil}{|S|}.$$

Proof. The network requires

$$N \leq s\beta^2 \leq \beta^{\lceil \log_\beta(n) \rceil + 1} \lceil \log_\beta(n) \rceil$$

random numbers since the radix-β switching network uses $s \leq \beta^{\lceil \log_\beta(n) \rceil} \lceil \log_\beta(n) \rceil / \beta$ switches of radix no greater than β by Theorem 2.2 on page 35, and each switch uses no more than β^2 random numbers as seen in the symbolic matrix (2.7) on the last page.

Let

$$\tilde{A} = \left(\prod_{k=1}^{s} \hat{\mathcal{E}}_k\right) A.$$

The matrix A is over the field $\mathbb{F}$, so each row of A is a row vector of polynomials in $\alpha_1, \ldots, \alpha_N$ of degree zero. Each switch in the generalized radix-β switching network increases the degree of the polynomials by one, and the depth of the network is $\lceil \log_\beta(n) \rceil$. This means the rows of $\tilde{A}$ are vectors of polynomials in $\alpha_1, \ldots, \alpha_N$ of degree $\lceil \log_\beta(n) \rceil$, and the determinant of an $r \times r$ submatrix of this preconditioned matrix is a polynomial

of degree $r\lceil \log_\beta(n) \rceil$.

Given that A has r linearly independent rows, we can designate these rows to be switched by the generalized radix-β switching network of Theorem 2.2 on page 35 to the first r rows of the preconditioned matrix $\tilde{A}$. This means there is an $r \times r$ submatrix of the first r rows of $\tilde{A}$ whose determinant is not identically zero. Because this is a polynomial of degree $r\lceil \log_\beta(n) \rceil$, the Schwartz/Zippel lemma tells us that $(a_1, \ldots, a_N)$ is a root of it with probability no greater than $r\lceil \log_\beta(n) \rceil / |S|$. With probability no less than $1 - r\lceil \log_\beta(n) \rceil / |S|$, it is not a root of the polynomial, and thus we have an $r \times r$ submatrix of the first r rows of $\tilde{A}$ whose determinant is not zero. Therefore, the first r rows of $\tilde{A}$ are linearly independent with probability no less than

$$1 - \frac{r\lceil \log_\beta(n) \rceil}{|S|} \geq 1 - \frac{n\lceil \log_\beta(n) \rceil}{|S|}. \qquad \blacksquare$$

The $\log_\beta(n)$ term in Theorem 2.3 means a preconditioner based on a generalized radix-β switching network offers some improvement as β increases. However, the number of random values required, and thus the size of the random set S from which they are chosen, grows exponentially for the particular exchange matrix (2.7). Further work should investigate better exchange matrices.

Chapter 3

Preconditioners and Determinantal Divisors

The random diagonal matrices in the preconditioners for computing the rank and determinant of a black box matrix presented in Sections 1.1.3 and 1.1.4, respectively, serve to ensure cyclicity of the nonzero eigenvalues of the preconditioned matrix, so the preconditioned matrix will have only one Jordan block for each nonzero eigenvalue. For the determinant problem, the preconditioned matrix $\tilde{A}$ must be nonsingular and cyclic for a nonsingular matrix A. Using the terminology of Chen *et al.* (2002, Section 3.1.3), the preconditioner for the determinant problem must solve the PreCondCyc problem. For the rank problem, the preconditioner must solve the PreCondCycNil problem. In other words, the preconditioned matrix $\tilde{A}$ must have the property that its minimal polynomial is $f^{\tilde{A}}(\lambda) = \lambda\, g(\lambda)$ and its characteristic polynomial is $\det(\lambda I - \tilde{A}) = \lambda^k g(\lambda)$

where $g(0) \neq 0$ and $k \geq 1$.

The notion of ensuring there is only one Jordan block for each nonzero eigenvalue for a matrix is an interesting one and is unique to symbolic linear algebra. In numerical linear algebra, one often tries to diagonalize a matrix, which often creates several Jordan blocks for a nonzero eigenvalue. For example, the identity matrix

$$I = \begin{bmatrix} 1 & & \\ & \ddots & \\ & & 1 \end{bmatrix}$$

is often the goal of a numerical preconditioner, but it is one of the worst matrices for the Wiedemann determinant algorithm since it has only a linear minimal polynomial, $f^I = \lambda - 1$.

The standard approach to proving a cyclicity preconditioner is to show that the characteristic polynomial $\det(\lambda I - \tilde{A})$ of the preconditioned matrix $\tilde{A}$ is squarefree, which means $\tilde{A}$ has n distinct eigenvalues. Wiedemann (1986, Section V) uses induction and a generic rank profile obtained with a parameterized matrix P to prove his preconditioner (1.8) is squarefree. Following his proof, Kaltofen and Pan (1992, Section 2) prove inductively that the two Toeplitz matrices of their preconditioner (1.11) also ensure the characteristic polynomial is squarefree. In Chen *et al.* (2002, Section 4), we break from these inductive proofs to show directly that the characteristic polynomial of a matrix preconditioned with a diagonal matrix (1.12) is squarefree. For their rank precondi-

tioner (1.9), Kaltofen and Saunders (1991, Section 2) use the generic rank profile and Wiedemann's induction proof to show the degree of the minimal polynomial of their preconditioned matrix $\tilde{A}$ is $\deg(f^{\tilde{A}}) = \operatorname{rank}(A) + 1$. In Chen *et al.* (2002, Section 5), we again follow the inductive proofs of Wiedemann (1986) and Kaltofen and Pan (1992) to prove three Toeplitz matrices (1.10) also produce the desired degree.

However, a matrix need not have a squarefree characteristic polynomial in order to be cyclic. For example, a Jordan block

$$J = \begin{bmatrix} a & 1 & & \\ & \ddots & \ddots & \\ & & a & 1 \\ & & & a \end{bmatrix}, \qquad a \neq 0$$

is cyclic, but its characteristic polynomial is not squarefree,

$$\det(\lambda I - J) = (\lambda - a)^n.$$

In this chapter, we present a new approach to prove a cyclicity preconditioner that involves computing the determinantal divisors of the characteristic matrix.

In Section 3.1, we use this approach to prove a fast new preconditioner for the determinant problem for a dense integer matrix described in Section 1.1.5. The dense integer

matrix determinant preconditioner is in the form of a unit upper bidiagonal multiplier,

$$\tilde{A} = A \begin{bmatrix} 1 & a_1 & & \\ & \ddots & \ddots & \\ & & 1 & a_{n-1} \\ & & & 1 \end{bmatrix}.$$

Then, in Section 3.2, we use the same approach to relax slightly the generic rank profile condition in the Kaltofen-Saunders rank algorithm described in Section 1.1.3.

3.1 Determinant-Preserving Preconditioners

To compute the determinant of a matrix A by Wiedemann's algorithm, we must precondition the matrix so that, if the original matrix A is nonsingular, the preconditioned matrix $\tilde{A}$ is nonsingular and cyclic. This cyclicity ensures the characteristic and minimal polynomials ($\det(\lambda I - \tilde{A})$ and $f^{\tilde{A}}$, respectively) agree. We can then use Wiedemann's algorithm to compute the minimal polynomial of the preconditioned matrix,

$$f^{\tilde{A}} = \lambda^n + f^{\tilde{A}}[n-1]\lambda^{n-1} + \cdots + f^{\tilde{A}}[0] = \det(\lambda I - \tilde{A}).$$

The determinant of $\tilde{A}$ is $\det(\tilde{A}) = (-1)^n f^{\tilde{A}}[0]$. If the preconditioner changes the matrix in a known way, we can recover the determinant of A.

One might think the requirement that A be nonsingular could pose a problem, but

this is not the case. If the matrix is singular, multiplication by any matrix will result in another singular matrix $\tilde{A}$. All singular matrices have a zero eigenvalue, and thus their minimal polynomials have zero constant terms

$$f^{\tilde{A}} = f^{\tilde{A}}[m]\lambda^m + f^{\tilde{A}}[m-1]\lambda^{m-1} + \cdots + f^{\tilde{A}}[1]\lambda.$$

Because the minimal polynomial divides the characteristic polynomial, a zero constant term in $f^{\tilde{A}}$ means $\det(\tilde{A}) = 0$. Thus, the constant coefficient of the minimal polynomial of every singular matrix must be zero.

Recall the definition of the determinantal divisors, invariant factors, and the Smith form of a polynomial matrix.

Definition 3.1. Let $\mathbb{F}$ be a field and $M \in \mathbb{F}^{n\times n}[\lambda]$. Then, the k-th determinantal divisor $s_k^*(M)$ of M is the greatest common divisor of all $k \times k$ minors of the matrix, and the Smith form of M is the diagonal matrix

$$S(M) = \operatorname{diag}(s_1(M), \ldots, s_r(M), 0, \ldots, 0) \in \mathbb{F}^{n\times n}[\lambda]$$

where $s_k(M)$ is the k-th invariant factor of M and is given by the equation

$$s_k(M) = \begin{cases} s_k^*(M) / s_{k-1}^*(M) & \text{if } k \neq 1 \\ s_k^*(M) & \text{if } k = 1. \end{cases}$$

Here, r is the maximal integer such that $s_r^*(M) \neq 0$ (Gohberg *et al.*, 1982, Chapter S1).

Notice $s_n^*(M) = \det(M)$, so for any square matrix A over a field, the largest determinantal divisor of its characteristics matrix is its characteristic polynomial,

$$s_n^*(\lambda I - A) = \det(\lambda I - A) \neq 0.$$

Thus, the Smith form of the characteristic matrix is nonsingular,

$$S(\lambda I - A) = \operatorname{diag}(s_1(\lambda I - A), \ldots, s_n(\lambda I - A)).$$

The largest invariant factor of $\lambda I - A$ is the minimal polynomial f^A of A. One way to prove that a matrix is cyclic is to show that all invariant factors of its characteristic matrix, except the largest, are one. We can do this by examining its determinantal divisors.

Lemma 3.1. *Let $\mathbb{F}$ be a field, $A \in \mathbb{F}^{n\times n}$, and*

$$\mathcal{U} = \begin{bmatrix} 1 & \alpha_1 & & \\ & \ddots & \ddots & \\ & & 1 & \alpha_{n-1} \\ & & & 1 \end{bmatrix} \in \mathbb{F}[\alpha_1, \alpha_2, \ldots, \alpha_{n-1}]^{n\times n},$$

where $\alpha_1, \ldots, \alpha_{n-1}$ are distinct indeterminates over $\mathbb{F}$. Let λ be an indeterminate distinct from $\alpha_1, \ldots, \alpha_{n-1}$. Then, for all k with $1 \leq k \leq n-1$, the greatest common divisor of

the $k \times k$ minors of $\lambda I - A\mathcal{U}$ in $\mathbb{F}[\alpha_1, \ldots, \alpha_{n-1}, \lambda]$ is contained in $\mathbb{F}[\lambda]$.

Proof. Let $A^{[j]}$ be the j-th column of the matrix A. Then,

$$A\mathcal{U} = \begin{bmatrix} A^{[1]} & A^{[2]} + \alpha_1 A^{[1]} & \cdots & A^{[n]} + \alpha_{n-1} A^{[n-1]} \end{bmatrix}$$

and

$$\lambda I - A\mathcal{U} = \begin{bmatrix} \lambda e_1 - A^{[1]} & \lambda e_2 - A^{[2]} - \alpha_1 A^{[1]} & \cdots & \lambda e_n - A^{[n]} - \alpha_{n-1} A^{[n-1]} \end{bmatrix},$$

where e_i is the i-th elementary unit vector and λ is an indeterminate in $\mathbb{F}$ distinct from $\alpha_1, \ldots, \alpha_{n-1}$. Each α_j occurs only in column $j+1$. Any $k \times k$ principal minor of $\lambda I - A\mathcal{U}$ that does not include the $(j+1)$-st row and column of the matrix is constant in α_j. These minors are also the characteristic polynomials of the corresponding submatrices of $A\mathcal{U}$, so they are not identically zero. ■

Similar arguments hold for $\mathcal{U}A$, $\mathcal{U}^{\mathrm{T}}A$, and $A\mathcal{U}^{\mathrm{T}}$.

Lemma 3.2. *Let $\mathbb{F}$ be a field, $A \in \mathbb{F}^{n \times n}$ have m of its first $n-1$ columns linearly independent, and*

$$\mathcal{U} = \begin{bmatrix} 1 & \alpha_1 & & \\ & \ddots & \ddots & \\ & & & \\ & & 1 & \alpha_{n-1} \\ & & & 1 \end{bmatrix} \in \mathbb{F}[\alpha_1, \alpha_2, \ldots, \alpha_{n-1}]^{n \times n},$$

where $\alpha_1, \ldots, \alpha_{n-1}$ are distinct indeterminates over $\mathbb{F}$. Let λ be an indeterminate distinct from $\alpha_1, \ldots, \alpha_{n-1}$. Then, the first m invariant factors of $\lambda I - A\mathcal{U}$ are all one.

Proof. Let $A^{[i_1,\ldots,i_l;j_1,\ldots,j_l]}$ be the submatrix of the matrix A formed by the intersection of rows $i_1, \ldots, i_l$ and columns $j_1, \ldots, j_l$, so $\det\left(A^{[i_1,\ldots,i_l;j_1,\ldots,j_l]}\right)$ is the corresponding minor. Because m of the first $n-1$ columns of A are linearly independent, for every k with $1 \leq k \leq m$, there is a $k \times k$ minor, denoted $\det\left(A^{[i_1,\ldots,i_k;j_1,\ldots,j_k]}\right)$, that is nonzero. The coefficient of the term $\prod_{l=1}^{k} \alpha_{j_l}$ in the minor $\det\left((\lambda I - A\mathcal{U})^{[i_1,\ldots,i_k;j_1+1,\ldots,j_k+1]}\right)$ is $(-1)^k \det\left(A^{[i_1,\ldots,i_k;j_1,\ldots,j_k]}\right) \neq 0$.

Consider the graded lexicographic ordering. Terms are ordered first by decreasing total degree. Terms of the same total degree are ordered lexicographically using the variable ordering

$$\alpha_1 \succ \alpha_2 \succ \cdots \succ \alpha_{n-1} \succ \lambda.$$

This ordering forces the leading term of one $k \times k$ minor of the characteristic matrix $\lambda I - A\mathcal{U}$ to be free of λ. The leading term of the greatest common divisor of all $k \times k$ minors must also be free of λ. Lemma 3.1 on page 46 says this greatest common divisor is free of $\alpha_1, \ldots, \alpha_{n-1}$, so the leading term must be a constant. Thus, the greatest common divisors, and also the invariant factors, are one.

More formally, because each entry of $\lambda I - A\mathcal{U}$ has total degree of at most one and each α_j occurs in only one column of $\lambda I - A\mathcal{U}$, each $k \times k$ minor of $\lambda I - A\mathcal{U}$ has total degree of at most k and must be, at most, linear in each α_j, and only $\alpha_{j_1}, \ldots, \alpha_{j_k}, \lambda$

appear in $\det\left((\lambda I - A\mathcal{U})^{[i_1,\ldots,i_k;j_1+1,\ldots,j_k+1]}\right)$. Thus, the leading term of this minor is

$$\operatorname{lt}\left(\det\left((\lambda I - A\mathcal{U})^{[i_1,\ldots,i_k;j_1+1,\ldots,j_k+1]}\right)\right) = (-1)^k \det\left(A^{[i_1,\ldots,i_k;j_1,\ldots,j_k]}\right) \prod_{l=1}^{k} \alpha_{j_l}.$$

Let g_k denote the greatest common divisor of the $k \times k$ minors of $\lambda I - A\mathcal{U}$ in $\mathbb{F}[\alpha_1, \ldots, \alpha_{n-1}, \lambda] = \mathbb{F}[\alpha_1, \ldots, \alpha_{n-1}][\lambda]$. Thus,

$$g_k \mid \det\left((\lambda I - A\mathcal{U})^{[i_1,\ldots,i_k;j_1+1,\ldots,j_k+1]}\right)$$

and

$$\operatorname{lt}(g_k) \mid \operatorname{lt}\left(\det\left((\lambda I - A\mathcal{U})^{[i_1,\ldots,i_k;j_1+1,\ldots,j_k+1]}\right)\right). \tag{3.1}$$

By Lemma 3.1 on page 46, $g_k \in \mathbb{F}[\lambda]$, which means g_k is free of $\alpha_1, \ldots, \alpha_{n-1}$. The leading term of g_k is also free of $\alpha_1, \ldots, \alpha_{n-1}$. By Equation (3.1), the leading term of g_k is free of λ. Thus, the leading term must be a constant, $\operatorname{lt}(g_k) \in \mathbb{F}$. Therefore, $g_k \in \mathbb{F}$, and $g_k = 1$ because g_k must divide $\det(\lambda I - A\mathcal{U})$, which includes the monic term λ^n. By Gauß' Lemma, the greatest common divisor of the $k \times k$ minors of $\lambda I - A\mathcal{U}$ in $\mathbb{F}(\alpha_1, \ldots, \alpha_{n-1})[\lambda]$ must also be one. This, in turn, means the first m invariant factors of $\lambda I - A\mathcal{U}$ are all one. ■

Similar arguments hold for $A\mathcal{U}^{\mathrm{T}}$ when m of the last $n-1$ columns of A are linearly independent, $\mathcal{U}A$ when m of the last $n-1$ rows of A are linearly independent, and $\mathcal{U}^{\mathrm{T}}A$ when m of the first $n-1$ rows of A are linearly independent.

When A is nonsingular, its first $n-1$ columns are linearly independent, so all invariant factors of the characteristic matrix except the largest must be one. We now have a symbolic determinant-preserving preconditioner.

Theorem 3.1. *Let $\mathbb{F}$ be a field, $A \in \mathbb{F}^{n\times n}$ be nonsingular, and*

$$\mathcal{U} = \begin{bmatrix} 1 & \alpha_1 & & \\ & \ddots & \ddots & \\ & & 1 & \alpha_{n-1} \\ & & & 1 \end{bmatrix} \in \mathbb{F}[\alpha_1, \alpha_2, \ldots, \alpha_{n-1}]^{n\times n}$$

where $\alpha_1, \ldots, \alpha_{n-1}$ are distinct indeterminates over $\mathbb{F}$. Then, $A\mathcal{U}$ is nonsingular and cyclic.

Proof. Clearly, $\mathcal{U}$ is nonsingular, so $A\mathcal{U}$ is also.

By Lemma 3.2 on page 47, all invariant factors of $\lambda I - A\mathcal{U}$ except the largest must be one. The largest invariant factor, which is the minimal polynomial of $A\mathcal{U}$, must also be the characteristic polynomial, and thus the matrix is cyclic. ■

The comments following the proof of Lemma 3.2 on page 47 about $A\mathcal{U}^{\mathrm{T}}$, $\mathcal{U}A$, and $\mathcal{U}^{\mathrm{T}}A$ also apply to Theorem 3.1.

The symbolic preconditioner $\tilde{A} = A\mathcal{U}$ could be used in the baby steps/giant steps determinant algorithms. The algorithm would be deterministic, but it would also be exponential in the size of the input matrix. We can use the Schwartz-Zippel Lemma (Zippel, 1979; Schwartz, 1980; Zippel, 1990) to convert this symbolic preconditioner to

a randomized one that will make the baby steps/giant steps determinant algorithms probabilistic and polynomial time.

Theorem 3.2. *Let $\mathbb{F}$ be a field, $A \in \mathbb{F}^{n\times n}$ be nonsingular, and S be a finite subset of $\mathbb{F}$. If*

$$U = \begin{bmatrix} 1 & a_1 & & \\ & \ddots & \ddots & \\ & & 1 & a_{n-1} \\ & & & 1 \end{bmatrix}$$

where $a_1, \ldots, a_{n-1}$ are chosen uniformly and independently from S, then AU is nonsingular and cyclic with probability at least $1 - n(n-1)/(2\,|S|)$.

Proof. Suppose $|\mathbb{F}| > n(n-1)/2$; otherwise the result is trivial. Clearly U is nonsingular. Thus, AU is also nonsingular.

By Theorem 3.1 on the last page, if

$$\mathcal{U} = \begin{bmatrix} 1 & \alpha_1 & & \\ & \ddots & \ddots & \\ & & 1 & \alpha_{n-1} \\ & & & 1 \end{bmatrix} \in \mathbb{F}[\alpha_1, \alpha_2, \ldots, \alpha_{n-1}]^{n\times n}$$

where $\alpha_1, \ldots, \alpha_{n-1}$ are distinct indeterminates over $\mathbb{F}$, the matrix $A\mathcal{U}$ is nonsingular and

cyclic. Let $y \in \mathbb{F}^n$ be a vector such that

$$y, (A\mathcal{U})y, \ldots, (A\mathcal{U})^{n-1}y$$

are linearly independent. The determinant of the matrix with these vectors as its columns is a nonzero polynomial in $\alpha_1, \ldots, \alpha_{n-1}$. This polynomial has total degree at most $n(n-1)/2$ in the indeterminates $\alpha_1, \ldots, \alpha_{n-1}$. If values $a_1, \ldots, a_{n-1}$ for $\alpha_1, \ldots, \alpha_{n-1}$ are chosen uniformly and independently from S, the determinant is a nonzero element of $\mathbb{F}$ with probability at least $1 - n(n-1)/(2\,|S|)$ by the Schwartz-Zippel Lemma. In this case, if

$$U = \begin{bmatrix} 1 & a_1 & & \\ & \ddots & \ddots & \\ & & 1 & a_{n-1} \\ & & & 1 \end{bmatrix},$$

the vectors $y, (AU)y, \ldots, (AU)^{n-1}y$ are linearly independent. ■

Theorem 3.2 gives one preconditioner that preserves the determinant of the original matrix. This preconditioner consists of multiplying on the right by a random unit upper bi-diagonal matrix but there is nothing special about an upper bi-diagonal matrix or multiplication on the right. We could prove this by rewriting all of the preceding work to show the preconditioning may be attained through multiplication on either the right or the left by either a random unit upper bi-diagonal matrix or a random unit lower

bi-diagonal matrix, but the following corollaries prove all four cases give determinant-preserving preconditioners.

Corollary 3.1. *Let $\mathbb{F}$ be a field, $A \in \mathbb{F}^{n\times n}$ be nonsingular, and S be a finite subset of $\mathbb{F}$. If*

$$L = \begin{bmatrix} 1 & & & \\ a_1 & 1 & & \\ & \ddots & \ddots & \\ & & a_{n-1} & 1 \end{bmatrix}$$

where $a_1, \ldots, a_{n-1}$ are chosen uniformly and independently from S, then LA is nonsingular and cyclic with probability at least $1 - n(n-1)/(2\,|S|)$

Proof. Clearly L, LA, and A^{T} are all nonsingular, and $(LA)^{\mathrm{T}} = A^{\mathrm{T}}L^{\mathrm{T}}$ where L^{T} is of the form of the matrix U in Theorem 3.2 on page 51.

Let F be the Smith form of the matrix $\lambda I - A^{\mathrm{T}}L^{\mathrm{T}}$. There exist unimodular matrices P and Q such that $P(\lambda I - A^{\mathrm{T}}L^{\mathrm{T}})Q = F$. However, the Smith form of a matrix is symmetric, so

$$F = (P(\lambda I - A^{\mathrm{T}}L^{\mathrm{T}})Q)^{\mathrm{T}} = Q^{\mathrm{T}}(\lambda I - LA)P^{\mathrm{T}}.$$

Due to the uniqueness of the Smith form, the matrices $\lambda I - A^{\mathrm{T}}L^{\mathrm{T}}$ and $\lambda I - LA$ have the same Smith form, and, thus, $A^{\mathrm{T}}L^{\mathrm{T}}$ and LA have the same minimal and characteristic polynomials. Thus, LA is cyclic exactly when $A^{\mathrm{T}}L^{\mathrm{T}}$ is cyclic. ■

Corollary 3.1 uses transposition and the uniqueness of the Smith form to give a second determinant-preserving preconditioner for the baby steps/giant steps determinant

algorithms. The following lemma uses similarity transformations to convert these two preconditioners into two more determinant-preserving preconditioners. Thus, we have four possible determinant-preserving preconditioners for the baby steps/giant steps determinant algorithms.

Corollary 3.2. *Let $\mathbb{F}$ be a field, $A \in \mathbb{F}^{n\times n}$ be nonsingular, and S be a finite subset of $\mathbb{F}$. If*

$$L = \begin{bmatrix} 1 & & & \\ a_1 & 1 & & \\ & \ddots & \ddots & \\ & & a_{n-1} & 1 \end{bmatrix}$$

and

$$U = \begin{bmatrix} 1 & a_1 & & \\ & \ddots & \ddots & \\ & & 1 & a_{n-1} \\ & & & 1 \end{bmatrix}$$

where $a_1, \ldots, a_{n-1}$ are chosen uniformly and independently from S, then AL and UA are nonsingular and cyclic with probability at least $1 - n(n-1)/(2\,|S|)$.

Proof. Clearly L and U are nonsingular. Thus, AL and UA are also nonsingular. Also, $AL = L^{-1}(LA)L$ and $UA = U(AU)U^{-1}$. AL and UA are similar to LA and AU, respectively. Similar matrices have characteristic matrices with the same invariant factors (Horn and Johnson, 1985, page 154), and, thus, the matrices have the same minimal

and characteristic polynomials. Therefore, AL is cyclic if and only if LA is; similarly for UA and AU. ■

These four preconditioners all have determinant one. They use $n-1$ random field elements compared to n for the diagonal matrix of Chen *et al.* (2002, Section 4), and their probability of success is the same as for the diagonal matrix. They do, however, use $2n-2$ field operations instead of n.

3.2 Rank Preconditioner

To compute the rank of a matrix A using the Kaltofen-Saunders rank algorithm, we must precondition the matrix so the minimal polynomial of the preconditioned matrix $\tilde{A}$ is $f^{\tilde{A}}(\lambda) = \lambda\, g(\lambda)$ and its characteristic polynomial is $\det(\lambda I - \tilde{A}) = \lambda^k g(\lambda)$ where $g(0) \neq 0$. In other words, all of the invariant factors except the largest are one or λ. From the discussion of determinantal divisors and invariant factors at the beginning of Section 3.1, we see it is possible to find the degree of the minimal polynomial of a matrix by bounding it using the determinantal divisors of the characteristic matrix. We start this section very much like Section 3.1. In fact, our first lemma is almost identical to Lemma 3.1 on page 46.

Lemma 3.3. *Let $\mathbb{F}$ be a field, $A \in \mathbb{F}^{n\times n}$, and $\mathcal{D} = \operatorname{diag}(\delta_1, \ldots, \delta_n)$ where $\delta_1, \ldots, \delta_n$ are distinct indeterminates over $\mathbb{F}$. Let λ be an indeterminate distinct from $\delta_1, \ldots, \delta_n$. Then, for all k with $1 \leq k \leq n-1$, the greatest common divisor of the $k \times k$ minors of $\lambda I - A\mathcal{D}$ in $\mathbb{F}[\delta_1, \ldots, \delta_n, \lambda]$ is contained in $\mathbb{F}[\lambda]$.*

Proof. As in the proof of Lemma 3.1 on page 46, we can prove this lemma through the fact that each δ_j occurs in only one column. ■

A similar argument will also hold for $\mathcal{D}A$.

Continuing as in Section 3.1, we now show the first r determinantal divisors and invariant factors of the characteristic matrix $\lambda I - A\mathcal{D}$ are all constant. The following lemma is nearly identical to Lemma 3.2 on page 47.

Lemma 3.4. *Let $\mathbb{F}$ be a field, $A \in \mathbb{F}^{n\times n}$ have rank r with $r \leq n-1$, and $\mathcal{D} = \operatorname{diag}(\delta_1, \ldots, \delta_n)$ where $\delta_1, \ldots, \delta_n$ are distinct indeterminates over $\mathbb{F}$. Let λ be an indeterminate distinct from $\delta_1, \ldots, \delta_n$. Then, the first r determinantal divisors and invariant factors of $\lambda I - A\mathcal{D}$ are all one.*

Proof. Following the proof of Lemma 3.2 on page 47, the proof of this lemma uses a graded lexicographic ordering to show the first r determinantal divisors of $\lambda I - A\mathcal{D}$ must be constant. ■

Again, a similar argument will also hold for $\mathcal{D}A$.

We know the first r invariant factors of the characteristic matrix are constant–in fact one–but we don't know anything about the rest of them other than, except for the largest, they cannot depend on $\delta_1, \ldots, \delta_n$. In fact, these first r invariant factors are the only constant factors.

At this point, we diverge from the path of Section 3.1. In that section, we moved directly from Lemma 3.2 on page 47, which proves the first m invariant factors are all

one, to Theorem 3.1 on page 50, which proves the symbolic preconditioner was cyclic and, thus, suitable for the determinant problem. For the rank problem, we have one more lemma to consider before presenting the symbolic preconditioner. However, this lemma, unlike the preceding lemmas of this section, does not involve a preconditioning matrix.

Lemma 3.5. *Let $\mathbb{F}$ be a field and $A \in \mathbb{F}^{n\times n}$ have rank r with $r \leq n-1$. Let λ be an indeterminate over $\mathbb{F}$. Then, for all k with $r+1 \leq k \leq n$, the k-th invariant factor of $\lambda I - A$ is divisible by λ.*

Proof. Because $r = \operatorname{rank}(A)$, every $k \times k$ minor of A must be zero for all k with $k \geq r+1$; otherwise the rank of A would have to be greater than r. Consider any $k \times k$ minor of $\lambda I - A$, $\det\left((\lambda I - A)^{[i_1,\ldots,i_k;j_1,\ldots,j_k]}\right)$. Expanding this minor, the only term not containing a power of λ is

$$\det\left(A^{[i_1,\ldots,i_k;j_1,\ldots,j_k]}\right) = 0.$$

This is true for every $k \times k$ minor of $\lambda I - A$, so the the greatest common divisors of the $k \times k$ minors of $\lambda I - A$ in $\mathbb{F}[\lambda]$ must be divisible by λ. In particular, the $(r+1)$-st greatest common divisor, or the $(r+1)$-st determinantal divisor, of $\lambda I - A$ is divisible by λ. From Definition 3.1 on page 45, the $(r+1)$-st determinantal divisor $s^*_{r+1}(\lambda I - A)$ of $\lambda I - A$ is the product of the first $r+1$ invariant factors of $\lambda I - A$,

$$s^*_{r+1}(\lambda I - A) = \prod_{i=1}^{r+1} s_i(\lambda I - A).$$

Thus, there exists an i with $1 \leq i \leq r+1$ such that λ divides $s_i(\lambda I - A)$. Because the j-th invariant factor divides the $(j+1)$-st for all j, this means λ divides every invariant factor $s_k(\lambda I - A)$ with $k \geq i$. In particular, λ divides every invariant factor $s_k(\lambda I - A)$ with $k \geq r+1$. ■

Because $\text{rank}(A\mathcal{D}) \leq \text{rank}(A)$, Lemma 3.5 says all of the invariant factors of the characteristic matrix $\lambda I - A\mathcal{D}$, except the first r, are divisible by λ.

Corollary 3.3. *Let $\mathbb{F}$ be a field, $A \in \mathbb{F}^{n\times n}$ have rank r with $r \leq n-1$, and $\mathcal{D} = \text{diag}(\delta_1, \ldots, \delta_n)$ where $\delta_1, \ldots, \delta_n$ are distinct indeterminates over $\mathbb{F}$. Let λ be an indeterminate distinct from $\delta_1, \ldots, \delta_n$. Then, every invariant factor of $\lambda I - A\mathcal{D}$, except the first r, is divisible by λ.*

Proof. The proof follows from Lemma 3.4 on page 56, Lemma 3.5 on the last page, and the fact that $\text{rank}(A\mathcal{D}) \leq \text{rank}(A)$ by Sylvester's inequality (Gantmacher, 1977, page 66). ■

Once more, a similar argument is also true for $\mathcal{D}A$.

Lemma 3.3 on page 55, Lemma 3.4 on page 56, and Corollary 3.3 give an upper bound on the degree of the minimal polynomial. To find a lower bound, the matrix must have an additional property. In Kaltofen and Saunders (1991, Lemma 2), the additional property was that the leading principal minors of A are nonzero up to the rank of the matrix. However, this property is more restrictive than is required. The following theorem is a generalization of Kaltofen and Saunders's lemma.

Theorem 3.3. *Let $\mathbb{F}$ be a field, $A \in \mathbb{F}^{n\times n}$ have rank r with $r \leq n-1$, and $\mathcal{D} = \operatorname{diag}(\delta_1, \ldots, \delta_n)$ where $\delta_1, \ldots, \delta_n$ are distinct indeterminates over $\mathbb{F}$. If any $r \times r$ principal minor of A is nonzero, then the minimal polynomial of $A\mathcal{D}$ is $f^{A\mathcal{D}} = \lambda\, g(\lambda)$ where $g(0) \neq 0$ and has degree $\deg(f^{A\mathcal{D}}) = r+1$. Furthermore, the characteristic polynomial of $A\mathcal{D}$ is $\det(\lambda I - A\mathcal{D}) = \lambda^{n-r} g(\lambda)$.*

Proof. Let g be the polynomial such that $f^{A\mathcal{D}} = \lambda^k g(\lambda)$ where $g(0) \neq 0$. The proof of Lemma 4.1 in Chen *et al.* (2002) says the characteristic polynomial of $\mathcal{D}A$ has no repeated roots except λ. The same argument can be used to show the characteristic polynomial of $A\mathcal{D}$ also has no repeated roots except λ. This means all invariant factors of $\lambda I - A\mathcal{D}$ except the largest are either one or a power of λ, so

$$\det(\lambda I - A\mathcal{D}) = \lambda^p f^{A\mathcal{D}}.$$

Lemma 3.4 on page 56 says the first r invariant factors are one. At the same time, Lemma 3.3 on the last page says the other $n-r$ are all divisible by λ. Thus, the characteristic polynomial of $A\mathcal{D}$ is divisible by λ^{n-r}, and $p \geq n-r-1$. This gives an upper bound for the degree of the minimal polynomial, $\deg(f^{A\mathcal{D}}) \leq r+1$. However, there is an $r \times r$ principal minor $\det\left(A^{[i_1,\ldots,i_r;i_1,\ldots,i_r]}\right)$ of A that is nonzero. This means the determinant $\det(\lambda I - A\mathcal{D})$ contains the term

$$\pm \det\left(A^{[i_1,\ldots,i_r;i_1,\ldots,i_r]}\right) \left(\prod_{l=1}^{r} \delta_{i_l}\right) \lambda^{n-r} \neq 0,$$

which is not divisible by λ^{n-r+1}. Thus, the characteristic polynomial is not divisible by λ^{n-r+1}. Because λ divides the minimal polynomial $f^{A\mathcal{D}}$, $p = n - r - 1$, $k = 1$, and the product of all but the largest invariant factor of $\lambda I - A\mathcal{D}$ is

$$s^*_{n-1}(\lambda I - A\mathcal{D}) = \prod_{i=1}^{n-1} s_i(\lambda I - A\mathcal{D}) = \lambda^{n-r-1}.$$

Therefore, the minimal polynomial has degree $\deg(f^{A\mathcal{D}}) = r + 1$. ■

The same is also true of $\mathcal{D}A$.

The symbolic preconditioner $A\mathcal{D}$ could be used in the Kaltofen-Saunders rank algorithm. The algorithm would be deterministic, but it would also be exponential in the size of the input matrix. Again, we can use the Schwartz-Zippel Lemma (Zippel, 1979; Schwartz, 1980; Zippel, 1990) to convert this symbolic preconditioner to a randomized one that will make the Kaltofen-Saunders rank algorithm probabilistic and polynomial time.

Theorem 3.4. *Let $\mathbb{F}$ be a field, $A \in \mathbb{F}^{n\times n}$ have rank r with $r \leq n-1$, and S be a finite subset of $\mathbb{F}$. Let A have a nonzero $r \times r$ principal minor. If $D = \mathrm{diag}(d_1, \ldots, d_n)$ where $d_1, \ldots, d_n$ are chosen uniformly and independently from S, then the characteristic polynomial of AD is $\det(\lambda I - AD) = \lambda^{n-r} g(\lambda)$ where $g(0) \neq 0$ and the minimal polynomial of AD is $f^{AD} = \lambda\, g(\lambda)$ and has degree $\deg(f^{AD}) = r + 1$, all with probability at least*

$$1 - \frac{r(r+1)}{2\,|S|} \geq 1 - \frac{n(n-1)}{2\,|S|}.$$

Proof. Suppose $|\mathbb{F}| > r(r+1)/2$; otherwise the result is trivial. By Theorem 3.3 on page 59, if $\mathcal{D} = \mathrm{diag}(\delta_1, \ldots, \delta_n)$ where $\delta_1, \ldots, \delta_n$ are distinct indeterminates over $\mathbb{F}$, $\det(\lambda I - A\mathcal{D}) = \lambda^{n-r} g(\lambda)$ and $f^{A\mathcal{D}} = \lambda\, g(\lambda)$ where $g(0) \neq 0$ and $\deg(f^{A\mathcal{D}}) = r+1$. Let $y \in \mathbb{F}^n$ be a vector such that

$$y, (A\mathcal{D})y, \ldots, (A\mathcal{D})^r y$$

are linearly independent. There is a $(r+1) \times (r+1)$ submatrix of the matrix with these vectors as its columns whose determinant is a nonzero polynomial in $\delta_1, \ldots, \delta_n$. This polynomial has total degree at most $r(r+1)/2$ in the indeterminates $\delta_1, \ldots, \delta_n$. If values $d_1, \ldots, d_n$ for $\delta_1, \ldots, \delta_n$ are chosen uniformly and independently from S, the determinant is a nonzero element of $\mathbb{F}$ with probability at least $1 - r(r+1)/(2\,|S|) \geq 1 - n(n-1)/(2\,|S|)$ by the Schwartz-Zippel Lemma. In this case, if $D = \mathrm{diag}(d_1, \ldots, d_n)$, then the vectors

$$y, (AD)y, \ldots, (AD)^r y$$

are linearly independent and the invariant factors of $\lambda I - AD$ are $f_1, \ldots, f_s$, where $\hat{f}_1, \ldots, \hat{f}_s$ are the invariant factors of $\lambda I - A\mathcal{D}$ and f_i is obtained from $\hat{f}_i$ by replacing the indeterminates $\delta_1, \ldots, \delta_n$ with the values $d_1, \ldots, d_n$, respectively. ∎

Theorem 3.5 gives the required preconditioner. We no longer need to mix both the rows and columns of the matrix before applying the diagonal matrix. We only need to mix one or the other.

There is nothing unique about multiplying on the right by a diagonal matrix; we

could also multiply on the left by a diagonal matrix. We could prove this by rewriting all of the preceding work for $\mathcal{D}A$ and DA, but the following corollary proves it with less work.

Corollary 3.4. *Let $\mathbb{F}$ be a field, $A \in \mathbb{F}^{n\times n}$ have rank r with $r \leq n-1$, and S be a finite subset of $\mathbb{F}$. Let A have a nonzero $r \times r$ principal minor. If $D = \operatorname{diag}(d_1, \ldots, d_n)$ where $d_1, \ldots, d_n$ are chosen uniformly and independently from S, then the minimal polynomial of DA is $f^{DA} = \lambda\, g(\lambda)$ where $g(0) \neq 0$ and has degree $\deg(f^{DA}) = r+1$ with probability at least*

$$1 - \frac{r(r+1)}{2\,|S|} \geq 1 - \frac{n(n-1)}{2\,|S|}.$$

Proof. Clearly, $(DA)^{\mathrm{T}} = A^{\mathrm{T}}D^{\mathrm{T}} = A^{\mathrm{T}}D$. Let F be the Smith form of the matrix $\lambda I - A^{\mathrm{T}}D$. There exist unimodular matrices P and Q such that $P(\lambda I - A^{\mathrm{T}}D)Q = F$. However, the Smith form of a matrix is symmetric, so

$$F = \left(P(\lambda I - A^{\mathrm{T}}D)Q\right)^{\mathrm{T}} = Q^{\mathrm{T}}(\lambda I - DA)P^{\mathrm{T}}.$$

Due to the uniqueness of the Smith form, the matrices $\lambda I - A^{\mathrm{T}}D$ and $\lambda I - DA$ have the same Smith form, and thus $A^{\mathrm{T}}D$ and DA have the same minimal and characteristic polynomials. ∎

This same proof also shows that the preconditioner DA of Chen *et al.* (2002, Section 4) could be replaced with AD.

The fact that we only need to mix rows or columns, and not both, may give an advan-

tage over the preconditioner for the Kaltofen-Saunders rank algorithm. A preconditioner based on a Beneš network suffices, but it would only be marginally better than using the generalized butterfly network preconditioners of Chen *et al.* (2002), since a Beneš network uses one fewer level of switches than two generalized butterfly networks. On the other hand, a preconditioner based on only one generalized butterfly network is not sufficient since it cannot ensure the existence a nonzero $r \times r$ principal minor when n is not a power of two. However, not being able to generalize a Beneš network to arbitrary size n may not be of great concern for the rank problem since embedding the original matrix into a $2^{\lceil \log_2 n \rceil} \times 2^{\lceil \log_2 n \rceil}$ matrix that is zero otherwise adds no new work for the switches of Chen *et al.* (2002).

Theorem 3.4 on page 60 and Corollary 3.4 on the previous page give Monte Carlo methods to compute the rank of a matrix A. There is currently no certificate for the rank of a matrix over an arbitrary field. However, these Monte Carlo methods will always return a value that is no larger than the rank of the matrix.

Theorem 3.5. *Let $\mathbb{F}$ be a field, $A \in \mathbb{F}^{n\times n}$ have rank r, and $D = \operatorname{diag}(d_1, \ldots, d_n) \in \mathbb{F}^{n\times n}$. Then, the characteristic polynomials of AD and DA are divisible by λ^{n-r} and the degree of the minimal polynomials of AD and DA are at most $\min\{n, r+1\}$.*

Proof. By Sylvester's inequality (Gantmacher, 1977, page 66), $\operatorname{rank}(AD) \leq \operatorname{rank}(A)$ and $\operatorname{rank}(DA) \leq \operatorname{rank}(A)$.

If A is nonsingular, $\operatorname{rank}(AD) \leq r = n$. Then, $n - r = 0$, so $\lambda^{n-r} = 1$, which always divides the characteristic polynomial $\det(\lambda I - AD)$. At the same time, the degree of the

minimal polynomial f^{AD} is

$$\deg(f^{AD}) \leq n = r < r + 1.$$

If, on the other hand, A is singular, $\text{rank}(AD) \leq r \leq n - 1$. Then, by Lemma 3.5 on page 57, the k-th invariant factor of $\lambda I - AD$ is divisible by λ for all k with $\text{rank}(AD)+1 \leq k \leq n$. In particular, the k-th invariant factor of $\lambda I - AD$ is divisible by λ for all k with $r + 1 \leq k \leq n$. Thus, λ^{n-r-1} divides the product of the first $n - 1$ invariant factors of $\lambda I - AD$,

$$\lambda^{n-r-1} \mid \prod_{k=1}^{n-1} s_k \left(\lambda I - AD\right)$$

and $\deg(\prod_{k=1}^{n-1} s_k \left(\lambda I - AD\right)) \geq n - r - 1$. Because $\det(\lambda I - AD)$ is the product of the n invariant factors of $\lambda I - AD$ and f^{AD} is the n-th invariant factor of $\lambda I - AD$, the degree of f^{AD} is

$$\begin{aligned}
\deg(f^{AD}) &= \deg(s_n \left(\lambda I - AD\right)) \\
&= \deg(\det(\lambda I - AD)) - \deg(\prod_{k=1}^{n-1} s_k \left(\lambda I - AD\right)) \\
&\leq n - (n - r - 1) = r + 1 \leq n.
\end{aligned}$$

A similar argument holds for $\deg(f^{DA})$. ■

Chapter 4

Block Wiedemann Method

The original Wiedemann method uses two vector projections, u and v, to produce a scalar sequence $\{u^{\mathrm{T}}A^i v\}_{i=0}^{\infty}$ that is linearly generated by the minimal polynomial f^A of the matrix A (Wiedemann, 1986). The block Wiedemann method introduced by Coppersmith (1994) uses two block projections, $X \in \mathbb{F}^{n\times\beta_1}$ and $Y \in \mathbb{F}^{n\times\beta_2}$ with $1 \leq \beta_1, \beta_2 \leq n$, to construct a block matrix sequence $\{X^{\mathrm{T}}A^i Y\}_{i=0}^{\infty}$. This matrix sequence is linearly generated by not only a scalar polynomial, but also by vector and matrix polynomials. It also has both a minimal generating polynomial and a minimal generating matrix polynomial.

To compute the minimal generating matrix polynomial, Kaltofen (1995) solves a homogeneous block Toeplitz system. Equivalently, one could solve a homogeneous block Hankel system. Coppersmith (1994) and Dickinson *et al.* (1974) use generalizations of the Berlekamp-Massey algorithm to the multivariate case. Dickinson *et al.* note that

Rissanen (1972) developed another multivariate algorithm, but that it is more a generalization of the approach in Rissanen (1971) than the Berlekamp-Massey algorithm since it uses the idea that a new solution should be a linear combination of previous solutions, but not the observation that the essential contained in all previous partial solutions is summarized by one particular past solution (Dickinson *et al.*, 1974, Section VII).

Beckermann and Labahn (1994) introduce the power Hermite-Padé approximation problem, which is a generalization of the Hermite-Padé approximation problem. They then follow the lead of Van Barel and Bultheel (1992) and give examples showing how other Hermite-Padé approximation problems can be stated in terms of a power Hermite-Padé approximation problem so that all may be solved in a unified manner. Beckermann and Labahn describe a Fast Power Hermite-Padé Solver (FPHPS) algorithm to solve their power Hermite-Padé problem by computing all solutions along a "diagonal path" (Beckermann and Labahn, 1994, Section 3). This computational technique was also used by Van Barel and Bultheel (1991) to solve the Hermite-Padé approximation problem.

In Section 4.1, we define the right minimal generating matrix polynomial F for a linearly generated matrix sequence $\{B_i\}_{i=0}^{\infty}$ and some properties of the sequence and F. Then, in Section 4.2, we discuss how to compute F using the Beckermann-Labahn Fast Power Hermite-Padé Solver (FPHPS) algorithm. In Section 4.3, we investigate additional properties of the block Wiedemann and block Krylov sequences and their right minimal generating matrix polynomials, $F_X^{A,Y}$ and $F^{A,Y}$, respectively. Finally, we present a block Monte Carlo algorithm for computing the rank of a matrix A in Section 4.4.

4.1 Linearly Generated Matrix Sequences

In this section, we discuss the notions of generating a sequence of matrices by vector and matrix polynomials. We then show that the set of all right generating vector polynomials of the sequence forms a module over the polynomials $\mathbb{F}[\lambda]$, and that this module has a basis over $\mathbb{F}[\lambda]$. We use this basis to define the right minimal generating matrix polynomial for the matrix sequence. Finally, we discuss how a right generating vector polynomial is related to an equivalence relation modulo a power of λ similar to the equivalence relation the Berlekamp-Massey algorithm solves (Dornstetter, 1987, Theorem 2). This equivalence relation can be used to compute the right minimal generating matrix polynomial.

Before we discuss what it means for a matrix sequence to be linearly generated by vector and matrix polynomials, we must first recall what it means for a nonzero polynomial to linearly generate a sequence in a vector space. The following definition comes from standard recursion theory. (See, for example, Kaltofen, 1995, Section 3, and Kaltofen, 1992a.)

Definition 4.1. Let $\mathbb{F}$ be a field, and let the matrix sequence $\{B_i\}_{i=0}^{\infty} \in \left(\mathbb{F}^{\beta_1 \times \beta_2}\right)^{\mathbb{Z}_{\geq 0}}$ with $\beta_1, \beta_2 > 0$. Then, the matrix sequence is linearly generated by the (scalar) polynomial $g = \sum_{i=0}^{d} g[i]\lambda^i \in \mathbb{F}[\lambda]$ if $g \neq 0$ and

$$\sum_{i=0}^{d} g[i] B_{i+j} = 0^{\beta_1 \times \beta_2}, \qquad \forall j \geq 0.$$

The polynomial g is a generating (scalar) polynomial for the matrix sequence.

By replacing the coefficients $g[i]$ Definition 4.1 on the previous page with vector coefficients $C[i]$ over the field, we get a similar definition for vector polynomials.

Definition 4.2. Let $\mathbb{F}$ be a field, and let the matrix sequence $\{B_i\}_{i=0}^{\infty} \in \left(\mathbb{F}^{\beta_1 \times \beta_2}\right)^{\mathbb{Z}_{\geq 0}}$ with $\beta_1, \beta_2 > 0$. Then, the matrix sequence is linearly generated from the right by the vector polynomial

$$C = \sum_{i=0}^{d} C[i]\lambda^i \in (\mathbb{F}[\lambda])^{\beta_2} = \mathbb{F}^{\beta_2}[\lambda]$$

if $C \neq 0$ and

$$\sum_{i=0}^{d} B_{i+j} C[i] = 0^{\beta_1}, \qquad \forall j \geq 0.$$

The vector polynomial C is a right generating vector polynomial for the matrix sequence (Kaltofen and Villard, 2001, Section 3).

We have defined a right generating vector polynomial for the sequence. Because vectors do not commute under multiplication with matrices, a vector polynomial that generates the sequence from the right will not generally generate it from the left. However, we can use a similar definition to define a left generating vector polynomial. In this dissertation, we will consider mostly right generating vector polynomials. For convenience, we will consider any generating vector polynomial for a matrix sequence to generate the sequence from the right unless otherwise specified.

We are now ready to take our first step towards defining the minimal generating matrix polynomial of a matrix sequence. The polynomial vectors of dimension β_2, $\mathbb{F}^{\beta_2}[\lambda]$, form

a module over the ring of polynomials $\mathbb{F}[\lambda]$. We begin by showing that the set of right generating vector polynomials for any matrix sequence $\{B_i\}_{i=0}^{\infty} \in \left(\mathbb{F}^{\beta_1 \times \beta_2}\right)^{\mathbb{Z}_{\geq 0}}$ forms a submodule of the module of vector polynomials $\mathbb{F}^{\beta_2}[\lambda]$ over the polynomials $\mathbb{F}[\lambda]$.

Lemma 4.1. *Let $\mathbb{F}$ be a field and $\beta_1, \beta_2 > 0$. The set of right generating vector polynomials of the matrix sequence $\{B_i\}_{i=0}^{\infty} \in \left(\mathbb{F}^{\beta_1 \times \beta_2}\right)^{\mathbb{Z}_{\geq 0}}$ forms a $\mathbb{F}[\lambda]$-submodule of the $\mathbb{F}[\lambda]$-module $\mathbb{F}^{\beta_2}[\lambda]$.*

Proof. Let $\mathbb{G}$ be the set of right generating vector polynomials for the matrix sequence. Clearly, $\mathbb{G} \subset \mathbb{F}^{\beta_2}[\lambda]$, so we only need to show $\mathbb{G}$ is closed under addition and polynomial multiplication.

Let $C_1 = \sum_{i=0}^{d_1} C_1[i]\lambda^i$ be a right generating vector polynomial for the matrix sequence. For any $g \in \mathbb{F}[\lambda]$,

$$\sum_{i=0}^{d_1} B_{j+i} g C_1[i] = g \sum_{i=0}^{d_1} B_{j+i} C_1[i] = 0^{\beta_1}, \qquad \forall j \geq 0,$$

which means $\mathbb{G}$ is closed under polynomial multiplication. Let $C_2 = \sum_{i=0}^{d_2} C_2[i]\lambda^i$ be another right generating vector polynomial for the matrix sequence, $C_3 = C_1 + C_2$, and $d_3 = \deg(C_3)$. Then,

$$\sum_{i=0}^{d_3} B_{j+i} C_3[i] = \sum_{i=0}^{d_1} B_{j+i} C_1[i] + \sum_{i=0}^{d_2} B_{j+i} C_2[i] = 0^{\beta_1}, \qquad \forall j \geq 0,$$

so $\mathbb{G}$ is closed under addition. ∎

Before we define the right minimal generating matrix polynomial for a matrix se-

quence, we must first describe what it means for a matrix sequence to be linearly generated by a matrix polynomial.

Definition 4.3. Let $\mathbb{F}$ be a field, and let the matrix sequence $\{B_i\}_{i=0}^{\infty} \in \left(\mathbb{F}^{\beta_1 \times \beta_2}\right)^{\mathbb{Z}_{\geq 0}}$ with $\beta_1, \beta_2 > 0$. Then, the matrix sequence is linearly generated from the right by the matrix polynomial

$$G = \sum_{i=0}^{d} G[i]\lambda^i \in \left(\mathbb{F}\left[\lambda\right]\right)^{\beta_1 \times \beta_2} = \mathbb{F}^{\beta_1 \times \beta_2}\left[\lambda\right]$$

if $\det(G) \neq 0$ and

$$\sum_{i=0}^{d} B_{i+j} G[i] = 0^{\beta_1 \times \beta_2}, \qquad \forall j \geq 0.$$

The matrix polynomial G is a right generating matrix polynomial for the sequence.

As one might expect, the columns of a right generating matrix polynomial are right generating vector polynomials for the matrix sequence. We can define a left generating matrix polynomial in a similar manner, and its rows will be left generating vector polynomials for the sequence. As with vector polynomials, we will consider only right generating matrix polynomials in this dissertation. For convenience, we will consider any generating matrix polynomial for a matrix sequence to generate the sequence from the right unless otherwise specified.

If the matrix polynomial G generates the matrix sequence $\{B_i\}_{i=0}^{\infty}$ from the right, its columns must be right generating vector polynomials of the sequence. Thus, any matrix sequence that is linearly generated from the right by a matrix polynomial must be linearly generated from the right by a vector polynomial.

By Lemma 4.1 on page 69, the right generating vector polynomials of the matrix sequence $\{B_i\}_{i=0}^{\infty}$ form a module over $\mathbb{F}[\lambda]$. This means any linear combination over $\mathbb{F}[\lambda]$ of right generating vector polynomials must also be a right generating vector polynomial. Thus, if the matrix polynomial G generates the matrix sequence, the columns of GM must also generate the sequence from the right for any matrix polynomial $M \in \mathbb{F}^{\beta_2 \times \beta_2}[\lambda]$. If $\det(M) \neq 0$, then $\det(GM) \neq 0$ and GM generates the sequence from the right.

Corollary 4.1 (to Lemma 4.1 on page 69). *Let $\mathbb{F}$ be a field, and let the matrix sequence $\{B_i\}_{i=0}^{\infty} \in \left(\mathbb{F}^{\beta_1 \times \beta_2}\right)^{\mathbb{Z}_{\geq 0}}$ with $\beta_1, \beta_2 > 0$ be linearly generated from the right by the matrix polynomial G. Then, for any nonsingular matrix polynomial $M \in \mathbb{F}^{\beta_2 \times \beta_2}[\lambda]$, the matrix polynomial GM also generates the matrix sequence.*

Proof. The proof follows from Lemma 4.1 on page 69 and the multiplicativity of the determinant. ■

Using Corollary 4.1, we can now show a matrix sequence is linearly generated by a polynomial if and only if it is generated from the right by a matrix polynomial.

Theorem 4.1. *Let $\mathbb{F}$ be a field, and let the matrix sequence $\{B_i\}_{i=0}^{\infty} \in \left(\mathbb{F}^{\beta_1 \times \beta_2}\right)^{\mathbb{Z}_{\geq 0}}$ with $\beta_1, \beta_2 > 0$. Then, the matrix sequence is linearly generated by a polynomial if and only if it is linearly generated from the right by a matrix polynomial.*

Proof. Suppose the polynomial $g = \sum_{i=0}^{d} g[i]\lambda^i$ generates $\{B_i\}_{i=0}^{\infty}$. Then, the matrix

polynomial gI has determinant $\det(gI) = g^{\beta_2} \neq 0$ and

$$\sum_{i=0}^{d} B_{i+j} \left(g[i]I\right) = \sum_{i=0}^{d} B_{i+j} g[i] = \sum_{i=0}^{d} g[i] B_{i+j} = 0^{\beta_1 \times \beta_2}, \qquad \forall j \geq 0.$$

Thus, the matrix sequence is linearly generated from the right by the matrix polynomial gI.

On the other hand, suppose the matrix polynomial G generates $\{B_i\}_{i=0}^{\infty}$, and let $\mathrm{adj}(G)$ be the adjoint matrix of G. If $\det(G) = g = \sum_{i=0}^{d} g[i]\lambda^i$, then

$$\det(G \ \mathrm{adj}(G)) = \det(\det(G)I) = \det(gI) = g^{\beta_2} \neq 0$$

and the matrix polynomial gI generates the matrix sequence from the right. Thus,

$$\sum_{i=0}^{d} g[i] B_{i+j} = \sum_{i=0}^{d} B_{i+j} g[i] = \sum_{i=0}^{d} B_{i+j} \left(g[i]I\right) = 0^{\beta_1 \times \beta_2}, \qquad \forall j \geq 0,$$

and g generates the matrix sequence. ■

Similarly, a matrix sequence is linearly generated by a matrix polynomial from the left if and only if it is linearly generated by a polynomial. Thus, the three notions are equivalent, and we say such a matrix sequence is linearly generated.

Unlike a vector space, a module does not generally have a basis. However, the modules formed by the right generating vector polynomials of a linearly generated matrix sequence $\{B_i\}_{i=0}^{\infty}$ do have a basis over the ring of polynomials $\mathbb{F}[\lambda]$. We can show such a basis

exists by first showing there exists a basis for the module over the rational functions $\mathbb{F}(\lambda)$. We then show a basis over $\mathbb{F}(\lambda)$ that is chosen so that the matrix whose columns are the basis vectors has minimal determinantal degree spans the module over $\mathbb{F}[\lambda]$ and is thus a basis for the module over $\mathbb{F}[\lambda]$. We will use this basis over $\mathbb{F}[\lambda]$ to define the minimal generating matrix polynomial for the matrix sequence.

Lemma 4.2. *Let $\mathbb{F}$ be a field, and let the matrix sequence $\{B_i\}_{i=0}^{\infty} \in \left(\mathbb{F}^{\beta_1 \times \beta_2}\right)^{\mathbb{Z}_{\geq 0}}$ with $\beta_1, \beta_2 > 0$ be linearly generated. Then, the $\mathbb{F}[\lambda]$-module $\mathbb{G}$ of the right generating vector polynomials of the matrix sequence has a basis of β_2 elements over $\mathbb{F}[\lambda]$.*

Proof. Because $\{B_i\}_{i=0}^{\infty}$ is linearly generated, there exists a polynomial $g = \sum_{i=0}^{d} g[i]\lambda^i$ such that

$$\sum_{i=0}^{d} B_{i+j} g[i] = 0^{\beta_1 \times \beta_2}, \qquad \forall j \geq 0.$$

Let $e_i \in \mathbb{F}^{\beta_2}$ be the i-th unit vector in the vector space $\mathbb{F}^{\beta_2}$. Then,

$$\sum_{i=0}^{d} B_{i+j} g[i] e_k = 0^{\beta_2}, \qquad \forall j \geq 0, 1 \leq k \leq \beta_1,$$

so the vectors $ge_1, \ldots, ge_{\beta_2} \in \mathbb{F}^{\beta_2}[\lambda]$ generate the matrix sequence from the right and are linearly independent over the field of rational functions $\mathbb{F}(\lambda)$. Thus, $ge_1, \ldots, ge_{\beta_2}$ is a basis for $\mathbb{G}$ over $\mathbb{F}(\lambda)$.

Let the vectors $b_1, \ldots, b_{\beta_2}$ be a basis for $\mathbb{G}$ over $\mathbb{F}(\lambda)$ such that the determinantal degree of the $\beta_2 \times \beta_2$ matrix formed using the basis vectors $b_1, \ldots, b_{\beta_2}$ as its columns is

minimal. In other words, given any basis $b'_1, \ldots, b'_{\beta_2}$ for $\mathbb{G}$ over $\mathbb{F}(\lambda)$,

$$\deg\left(\det\left(\begin{bmatrix} b_1 & \cdots & b_{\beta_2} \end{bmatrix}\right)\right) \leq \deg\left(\det\left(\begin{bmatrix} b'_1 & \cdots & b'_{\beta_2} \end{bmatrix}\right)\right). \tag{4.1}$$

Suppose there exists a generating vector polynomial $C \in \mathbb{G}$ that is not a linear combination of $b_1, \ldots, b_{\beta_2}$ over $\mathbb{F}[\lambda]$. C is, however, a linear combination of $b_1, \ldots, b_{\beta_2}$ over $\mathbb{F}(\lambda)$. Thus,

$$C(\lambda) = \sum_{i=1}^{\beta_2} c_i(\lambda) b_i(\lambda),$$

where $c_i(\lambda) \notin \mathbb{F}[\lambda]$ for at least one i. Let $d(\lambda)$ be the least common denominator of all the c_i, and let $c_i = \overline{c}_i(\lambda) + r_i(\lambda)/d(\lambda)$ with $\deg(r_i) < \deg(d)$ for $1 \leq i \leq \beta_2$. Thus, $r_i(\lambda) \neq 0$ for at least one i. Let i_0 be given such that $r_{i_0}(\lambda) \neq 0$. The generating vector polynomial C is

$$C(\lambda) = \sum_{i=1}^{\beta_2} \left(\overline{c}_i(\lambda) + \frac{r_i(\lambda)}{d(\lambda)}\right) b_i(\lambda) = \sum_{i=1}^{\beta_2} \overline{c}_i(\lambda) b_i(\lambda) + \sum_{i=1}^{\beta_2} \frac{r_i(\lambda)}{d(\lambda)} b_i(\lambda).$$

Then,

$$C(\lambda) - \sum_{i=1}^{\beta_2} \overline{c}_i(\lambda) b_i(\lambda) = \sum_{i=1}^{\beta_2} \frac{r_i(\lambda)}{d(\lambda)} b_i(\lambda)$$

must be a vector polynomial, and the vector polynomials $b'_1, \ldots, b'_{\beta_2}$ where

$$b'_i = \begin{cases} \sum_{i=1}^{\beta_2} \left(r_i(\lambda)/d(\lambda)\right) b_i(\lambda) & \text{if } i = i_0, \\ b_i & \text{if } i \neq i_0. \end{cases}$$

form a basis for $\mathbb{G}$ over $\mathbb{F}(\lambda)$. However,

$$\det\left(\begin{bmatrix} b'_1 & \cdots & b'_{\beta_2} \end{bmatrix}\right) = \frac{r_{i_0}(\lambda)}{d(\lambda)} \det\left(\begin{bmatrix} b_1 & \cdots & b_{\beta_2} \end{bmatrix}\right),$$

so

$$\deg\left(\det\left(\begin{bmatrix} b'_1 & \cdots & b'_{\beta_2} \end{bmatrix}\right)\right) < \deg\left(\det\left(\begin{bmatrix} b_1 & \cdots & b_{\beta_2} \end{bmatrix}\right)\right),$$

which contradicts the minimal determinantal degree requirement (4.1) on the basis vectors $b_1, \ldots, b_{\beta_2}$. Thus, $b_1, \ldots, b_{\beta_2}$ must be a basis for $\mathbb{G}$ over $\mathbb{F}[\lambda]$. ■

Now that the module $\mathbb{G}$ has a basis over $\mathbb{F}[\lambda]$, we begin to see the definition of the right minimal generating matrix polynomial. Any matrix G whose columns form a basis over $\mathbb{F}[\lambda]$ for the module generates the matrix sequence from the right. However, just as a linearly generated scalar sequence may have multiple generating polynomials of minimal degree, there may be multiple right generating matrix polynomials whose columns form a basis over $\mathbb{F}[\lambda]$ for the module. We need a way to define a unique right minimal generating matrix polynomial. In the case of generating polynomials, the minimal polynomial is the monic generating polynomial of minimal degree. For matrix polynomials, we can show that the matrices formed by the basis elements of two basis over $\mathbb{F}[\lambda]$ for the module are right equivalent with respect to multiplication on the right by a unimodular matrix. This equivalence will allow us to use a canonical form to define the right minimal generating matrix polynomial of the matrix sequence.

Theorem 4.2. *Let $\mathbb{F}$ be a field, and let the matrix sequence $\{B_i\}_{i=0}^{\infty} \in \left(\mathbb{F}^{\beta_1 \times \beta_2}\right)^{\mathbb{Z}_{\geq 0}}$ with*

$\beta_1, \beta_2 > 0$ be linearly generated. Then, given any two bases over $\mathbb{F}[\lambda]$ for the module $\mathbb{G}$ of right generating vector polynomials of the matrix sequence, the matrices whose columns are the basis elements are right equivalent with respect to multiplication on the right by a unimodular matrix.

Proof. Let $b_1, \ldots, b_{\beta_2}$ and $b'_1, \ldots, b'_{\beta_2}$ be two bases for $\mathbb{G}$ over $\mathbb{F}[\lambda]$, and let

$$G_1 = \begin{bmatrix} b_1 & \ldots & b_{\beta_2} \end{bmatrix} \quad \text{and} \quad G_2 = \begin{bmatrix} b'_1 & \ldots & b'_{\beta_2} \end{bmatrix}.$$

The columns of G_2 must be linear combinations over $\mathbb{F}[\lambda]$ of the columns of G_1, so there exists a matrix polynomial $U_1 \in \mathbb{F}^{\beta_2 \times \beta_2}[\lambda]$ such that $G_2 = G_1 U_1$. Conversely, the columns of G_1 are linear combinations of the columns of G_2 over $\mathbb{F}[\lambda]$, so there exists another matrix polynomial $U_2 \in \mathbb{F}^{\beta_2 \times \beta_2}[\lambda]$ such that $G_1 = G_2 U_2$. Thus,

$$G_1 = G_2 U_2 = G_1 U_1 U_2$$

and $\det(U_1 U_2) = 1$, so U_1 and U_2 must be unimodular matrices with

$$\det(U_2) = \frac{1}{\det(U_1)}.$$ ∎

Theorem 4.2 says the determinants of any two matrices whose columns are the elements of two bases for $\mathbb{G}$ over $\mathbb{F}[\lambda]$ are equal up to a constant factor.

Corollary 4.2. *Let $\mathbb{F}$ be a field, and let the matrix sequence $\{B_i\}_{i=0}^{\infty} \in \left(\mathbb{F}^{\beta_1 \times \beta_2}\right)^{\mathbb{Z}_{\geq 0}}$ Let*

$b_1, \ldots, b_{\beta_2}$ and $b'_1, \ldots, b'_{\beta_2}$ be two bases for $\mathbb{G}$ over $\mathbb{F}[\lambda]$, and let

$$G_1 = \begin{bmatrix} b_1 & \cdots & b_{\beta_2} \end{bmatrix} \quad \text{and} \quad G_2 = \begin{bmatrix} b'_1 & \cdots & b'_{\beta_2} \end{bmatrix}.$$

Then, there exists a field element $c \in \mathbb{F}$ such that $\det(G_1) = c \det(G_2)$.

Proof. The proof follows from Theorem 4.2 on page 75. ■

For a canonical form of the right equivalence with respect to multiplication on the right by a unimodular matrix, recall the definition of the Popov form (Popov, 1972; Villard, 1997a, Definition 2).

Definition 4.4. Let $\mathbb{F}$ be a field, let $M \in \mathbb{F}^{\beta_2 \times \beta_2}[\lambda]$, and let $M^{[j]}$ be the j-th column of M. M is said to be column reduced if the matrix $[M]_c$ formed by the leading coefficients of the columns of M has rank equal to that of M, $\text{rank}([M]_c) = \text{rank}(M)$. Thus, the determinantal degree of M is the sum of the degrees of its columns,

$$\deg(\det(M)) = \sum_{j=1}^{n} \deg(M^{[j]}).$$

If, in addition, M satisfies the following properties, we say M is in Popov form:

1. the column degrees are increasingly ordered;
2. the last entry of degree $\deg(M^{[j]})$ in $M^{[j]}$ is monic and is called the pivot element of column j with row index r_j;
3. if $\deg(M^{[j]}) = \deg(M^{[k]})$ and $j < k$, then $r_j < r_k$;

4. all entries in a row containing a pivot element have degrees lower than that of the pivot element.

Any two matrices that are right equivalent with respect to multiplication on the right by a unimodular matrix have the same unique Popov form (Kailath, 1980, page 484). This canonical form, which is also called the polynomial echelon form and was first introduced by Popov (1970), can be used to define a unique right minimal generating matrix polynomial F for the linearly generated matrix sequence $\{B_i\}_{i=0}^{\infty}$.

Definition 4.5. Let $\mathbb{F}$ be a field, and let the matrix sequence $\{B_i\}_{i=0}^{\infty} \in \left(\mathbb{F}^{\beta_1 \times \beta_2}\right)^{\mathbb{Z}_{\geq 0}}$ with $\beta_1, \beta_2 > 0$ be linearly generated. Then, the right minimal generating matrix polynomial F for the linearly generated matrix sequence $\{B_i\}_{i=0}^{\infty}$ is the Popov canonical form of the matrices whose columns are the basis elements over $\mathbb{F}[\lambda]$ of the module $\mathbb{G}$ of right generating vector polynomials of the matrix sequence.

Again, there is a similar definition for the left minimal generating matrix polynomial, but we will constrain ourselves to only discussing the right minimal generating matrix polynomial unless otherwise specified.

One consequence of the minimal generating matrix polynomial being in Popov form is that it is column reduced. Because the columns of F form a basis over $\mathbb{F}[\lambda]$ for the module $\mathbb{G}$ of right generating vector polynomials of $\{B_i\}_{i=0}^{\infty}$, they must also form a basis over $\mathbb{F}(\lambda)$ for $\mathbb{G}$ and must be linearly independent over $\mathbb{F}[\lambda]$ and $\mathbb{F}(\lambda)$. Thus, both F and $[F]_c$ must have full rank. In other words, the leading coefficient vectors of the columns of F are linearly independent over $\mathbb{F}$. We can then show that there is no cancellation

in leading terms when multiplying F by a polynomial vector on the right. We first show there is no cancellation in the leading terms for any matrix M where the leading coefficient vectors of the columns of M are linearly independent over $\mathbb{F}$.

Lemma 4.3. *Let $\mathbb{F}$ be a field, let $M \in \mathbb{F}^{\beta_2 \times \beta_2}$, and let $M^{[i]}$ be the i-th column of M. Suppose the leading coefficient vectors of the columns of M are linearly independent over $\mathbb{F}$. Then, for any nonzero vector polynomial $C = \sum_{i=0}^{d} C[i]\lambda^i \in \mathbb{F}^{\beta_2}[\lambda]$, the degree of MC is*

$$\deg(MC) = \max_{1 \le i \le \beta_2} \{\deg(M^{[i]}) + \deg(C^{[i]})\}.$$

Proof. Let $M^{[i,j]}$ be the (i,j)-th entry of M and d be

$$d = \max_{1 \le i,j \le \beta_2} \{\deg(M^{[i,j]}) + \deg(C^{[j]})\} = \max_{1 \le j \le \beta_2} \{\deg(M^{[j]}) + \deg(C^{[j]})\}.$$

The i-th entry of MC is

$$(MC)^{[i]} = \sum_{j=1}^{\beta_2} M^{[i,j]} C^{[j]}$$

and has degree

$$\deg((MC)^{[i]}) \le \sum_{j=1}^{\beta_2} \deg(M^{[i,j]} C^{[j]}) = \sum_{j=1}^{\beta_2} \left(\deg(M^{[i,j]}) + \deg(C^{[j]})\right) \le d.$$

Thus, the degree of MC is bounded by

$$\deg(MC) = \max_{1 \le i \le \beta_2} \deg((MC)^{[i]}) \le d.$$

On the other hand,

$$MC = \sum_{j=1}^{\beta_2} M^{[j]}C^{[j]},$$

and the degree of $M^{[j]}C^{[j]}$ is bounded by

$$\deg(M^{[j]}C^{[j]}) = \max_{1 \le i \le \beta_2} \deg(M^{[i,j]}C^{[j]}) \le d.$$

By the definition of d, there must be at least one index j where this bound is attained. If it is attained at the index j, the coefficient of λ^d in $M^{[j]}C^{[j]}$ is the leading coefficient of $M^{[j]}C^{[j]}$; otherwise, the coefficient is zero. Thus, the coefficient of λ^d in $M^{[j]}C^{[j]}$ is

$$(M^{[j]}C^{[j]})[d] = \begin{cases} 0^{\beta_2}, & \text{if } \deg(M^{[j]}C^{[j]}) < d, \\ \operatorname{lc}(M^{[j]}) \operatorname{lc}(C^{[j]}) & \text{if } \deg(M^{[j]}C^{[j]}) = d, \end{cases}$$

for $1 \le j \le \beta_2$, and the coefficient of λ^d in MC is

$$(MC)[d] = \sum_{j=1}^{\beta_2} (M^{[j]}C^{[j]})[d] = \sum_{\substack{1 \le j \le \beta_2 \\ \deg(M^{[j]}C^{[j]}) = d}} \operatorname{lc}(M^{[j]}) \operatorname{lc}(C^{[j]}) \neq 0^{\beta_2}$$

since $(MC)[d]$ is a linear combination over $\mathbb{F}$ of the leading vector coefficients of the columns of M and these leading vector coefficients are linearly independent $\mathbb{F}$. This sum is not empty since there must exist at least one j such that $\deg(M^{[j]}C^{[j]}) = d$. ■

Lemma 4.3 gives the following corollary that says there is no cancellation in leading terms when multiplying F by a polynomial vector on the right.

Corollary 4.3. *Let $\mathbb{F}$ be a field, and let the matrix sequence $\{B_i\}_{i=0}^{\infty} \in \left(\mathbb{F}^{\beta_1 \times \beta_2}\right)^{\mathbb{Z}_{\geq 0}}$ with $\beta_1, \beta_2 > 0$ be linearly generated. Let F be the minimal generating matrix polynomial of $\{B_i\}_{i=0}^{\infty}$. Then, for any nonzero $C \in \mathbb{F}^{\beta_2}[\lambda]$,*

$$\deg(FC) = \max_{1 \leq i \leq \beta_2} \{\deg(F^{[i]}) + \deg(C^{[i]})\}.$$

Proof. The proof follows from Definition 4.5 on page 78 and Lemma 4.3 on page 79. ■

Corollary 4.3 also says the columns of F form a minimal basis for $\mathbb{G}$ (Villard, 1997a, Theorem 2; Forney, 1975, Main Theorem).

Corollary 4.1 on page 71 says that multiplying the minimal generating matrix polynomial on the right by a nonsingular matrix gives a matrix polynomial that generates the matrix sequence from the right. However, because the columns of the minimal generating matrix polynomial form a basis for the module of right generating vector polynomials, it is also possible to show that any generating matrix polynomial is the product of the minimal generating matrix polynomial and a nonsingular matrix polynomial.

Theorem 4.3. *Let $\mathbb{F}$ be a field, and let the matrix sequence $\{B_i\}_{i=0}^{\infty} \in \left(\mathbb{F}^{\beta_1 \times \beta_2}\right)^{\mathbb{Z}_{\geq 0}}$ with $\beta_1, \beta_2 > 0$ be linearly generated. Let F be the minimal generating matrix polynomial of $\{B_i\}_{i=0}^{\infty}$. If the matrix polynomial G generates $\{B_i\}_{i=0}^{\infty}$, then there exists a nonsingular matrix polynomial $M \in \mathbb{F}^{\beta_2 \times \beta_2}[\lambda]$ such that $G = FM$. Furthermore,* $\deg(\det(G)) \geq \deg(\det(F))$ *and* $\deg(G) \geq \deg(F)$.

Proof. By Definition 4.5 on page 78, the columns of F form a basis over $\mathbb{F}[\lambda]$ for $\mathbb{G}$.

Thus, there exists a matrix polynomial M such that $G = FM$. By the multiplicativity of the determinant, $\det(G) = \det(F)\det(M)$. However, $\det(G) \neq 0$ and $\det(F) \neq 0$ because the columns of the two matrix polynomials must be linearly independent over $\mathbb{F}[\lambda]$ and $\mathbb{F}(\lambda)$. Therefore, $\det(M) \neq 0$ and

$$\deg(\det(G)) = \deg(\det(F)) + \deg(\det(M)) \geq \deg(\det(F)).$$

Let $F^{[i]}$ be the i-th column of F and $G^{[i]}$ be the i-th column of G. There exists an i_0 such that the i_0-th column of F has the same degree as a vector polynomial as the matrix polynomial. By Corollary 4.3 on the last page, the degree of the j-th column of G is

$$\deg(G^{[j]}) = \max_{1 \leq i \leq \beta_2} \{\deg(F^{[i]}) + \deg(M^{[i,j]})\} \geq \deg(F^{[i_0]}) + \deg(M^{[i_0,j]})$$

for all $1 \leq j \leq \beta_2$, where $M^{[i,j]}$ is the (i,j)-th entry of M. If $\deg(G) < \deg(F)$, then

$$\deg(G^{[j]}) \leq \deg(G) < \deg(F) = \deg(F^{[i_0]}), \qquad 1 \leq j \leq \beta_2,$$

This means $\deg(M^{[i_0,j]}) < 0$ and $M^{[i_0,j]} = 0$ for $1 \leq j \leq \beta_2$. Thus, $\det(M) = 0$, which contradicts the first part of the proof. ■

We now turn our attention to some properties of the minimal generating matrix polynomial F that will allow us to compute it. We first look at the its degree as a matrix polynomial. The following corollary shows that this degree is less than the degree of the

minimal polynomial of the sequence.

Corollary 4.4. *Let $\mathbb{F}$ be a field, and let the matrix sequence $\{B_i\}_{i=0}^{\infty} \in \left(\mathbb{F}^{\beta_1 \times \beta_2}\right)^{\mathbb{Z}_{\geq 0}}$ with $\beta_1, \beta_2 > 0$ be linearly generated by the polynomial g. Then,* $\deg(F) \leq \deg(g)$.

Proof. Because g generates the matrix sequence, the matrix polynomial gI generates the sequence as well. By Theorem 4.3 on page 81, $\deg(F) \leq \deg(gI) = \deg(g)$. ■

Thus, the degree of the minimal generating matrix polynomial is no greater than the degree of the minimal polynomial of the matrix sequence. Let us define γ_2 to be the degree of the minimal generating matrix polynomial.

Definition 4.6. Let $\mathbb{F}$ be a field, and let the matrix sequence $\{B_i\}_{i=0}^{\infty} \in \left(\mathbb{F}^{\beta_1 \times \beta_2}\right)^{\mathbb{Z}_{\geq 0}}$ with $\beta_1, \beta_2 > 0$ be linearly generated. Then, γ_2 is the degree of the minimal generating matrix polynomial F for the linearly generated matrix sequence $\{B_i\}_{i=0}^{\infty}$, $\gamma_2 = \deg(F)$.

Example 4.1. Consider the matrix sequence generated by the polynomial $f = \lambda^2 - \lambda - 1$ with

$$B_0 = \begin{bmatrix} 0 & 1 \\ 1 & 2 \end{bmatrix} \text{ and } B_1 = \begin{bmatrix} 1 & 0 \\ 3 & 1 \end{bmatrix}.$$

Here, $\gamma_2 \leq 2$.

We now show the first γ_2 elements of the matrix sequence $\{B_i\}_{i=0}^{\gamma_2 - 1}$ generate the entire sequence $\{B_i\}_{i=0}^{\infty}$.

Lemma 4.4. *Let $\mathbb{F}$ be a field, and let the matrix sequence $\{B_i\}_{i=0}^{\infty} \in \left(\mathbb{F}^{\beta_1 \times \beta_2}\right)^{\mathbb{Z}_{\geq 0}}$ with $\beta_1, \beta_2 > 0$ be linearly generated. Then, there exists a matrix polynomial $\widetilde{F} = \sum_{i=0}^{\gamma_2 - 1} \widetilde{F}[i]\lambda^i$*

of degree at most $\gamma_2 - 1$ *such that*

$$B_{j+\gamma_2} = \sum_{i=0}^{\gamma_2-1} B_{i+j}\widetilde{F}[i], \qquad \forall j \geq 0.$$

Proof. Let $F^{[i]}$ be the i-th column of the minimal generating matrix polynomial F of the matrix sequence,

$$F = \left[F^{[1]} \quad \dots \quad F^{[\beta_2]} \right],$$

and let d_i be the degree of $F^{[i]}$, $d_i = \deg(F^{[i]})$. Thus, $d_i \leq \gamma_2$ for $1 \leq i \leq \gamma_2$ by Definition 4.6 on the last page. Consider the matrix polynomial

$$G = \left[\lambda^{\gamma_2 - d_1} F^{[1]} \quad \dots \quad \lambda^{\gamma_2 - d_{\gamma_2}} F^{[\gamma_2]} \right] = \sum_{i=0}^{\gamma_2} G[i]\lambda^i.$$

Every column of G generates the matrix sequence $\{B_i\}_{i=0}^{\infty}$ from the right, so

$$\sum_{i=0}^{\gamma_2} B_{i+j}G[i] = 0^{\beta_1 \times \beta_2}, \qquad \forall j \geq 0,$$

and

$$B_{j+\gamma_2}G[\gamma_2] = -\sum_{i=0}^{\gamma_2-1} B_{i+j}G[i], \qquad \forall j \geq 0.$$

The leading matrix coefficient of G is the matrix formed by the leading coefficients of the columns of the minimal generating matrix polynomial F, $G[\gamma_2] = [F]_c$. Because F is in Popov form, the matrix $G[\gamma_2]$ must have full rank and thus be invertible. Therefore, for

all $j \geq 0$,

$$B_{j+\gamma_2} = -\left(\sum_{i=0}^{\gamma_2-1} B_{i+j} G[i]\right) G[\gamma_2]^{-1} = \sum_{i=0}^{\gamma_2-1} B_{i+j} \left(-G[i]\, G[\gamma_2]^{-1}\right). \qquad \blacksquare$$

When $\beta_1 = \beta_2 = 1$, the linearly generated matrix sequence $\{B_i\}_{i=0}^{\infty}$ is a linearly generated sequence of field elements, $\{B_i\}_{i=0}^{\infty} \in \mathbb{F}^{\mathbb{Z}_{\geq 0}}$, and the minimal generating matrix polynomial F is the minimal polynomial f of the sequence, $F = f$. In this case, a generating vector polynomial for the sequence is just a polynomial over the field, and any polynomial that generates the first $2\gamma_2$ elements of the sequence must generate the entire sequence $\{B_i\}_{i=0}^{\infty}$ (Kaltofen, 1992a). The proof follows from showing the rank of the Hankel matrix $H(\gamma_2, \gamma_2 + 1)$ is maximal, where

$$H(\nu_1, \nu_2) = \begin{bmatrix} B_0 & B_1 & \cdots & B_{\nu_2-1} \\ B_1 & B_2 & \cdots & B_{\nu_2} \\ \vdots & \vdots & \ddots & \vdots \\ B_{\nu_1-1} & B_{\nu_1} & \cdots & B_{\nu_1+\nu_1-2} \end{bmatrix} \in \mathbb{F}^{\beta_1\nu_1 \times \beta_2\nu_2}.$$

More sequence elements may be needed to determine if a vector polynomial generates a matrix sequence, but the proof of how many elements are needed still involves rank considerations. However, for blocking factors greater than one, the matrix $H(\gamma_2, \gamma_2 + 1)$ is not a Hankel matrix, but a block Hankel matrix. Again, γ_2 plays a role.

Lemma 4.5. *Let $\mathbb{F}$ be a field, and let the matrix sequence $\{B_i\}_{i=0}^{\infty} \in \left(\mathbb{F}^{\beta_1 \times \beta_2}\right)^{\mathbb{Z}_{\geq 0}}$ with*

$\beta_1, \beta_2 > 0$ be linearly generated. Let $\nu_1 \geq 1$ and $\nu_2 \geq \gamma_2$. Then,

$$\operatorname{rank}(H(\nu_1, \nu_2)) = \operatorname{rank}(H(\nu_1, \gamma_2)).$$

Proof. We prove the lemma by induction on ν_2.

Suppose the lemma is true for $\nu_2 \geq \gamma_2$. Then, $\operatorname{rank}(H(\nu_1, \nu_2 + 1)) \geq \operatorname{rank}(H(\nu_1, \nu_2))$ because $H(\nu_1, \nu_2)$ is a submatrix of $H(\nu_1, \nu_2 + 1)$. By Lemma 4.4 on page 83, there exists a polynomial $\widetilde{F} = \sum_{i=0}^{\gamma_2 - 1} \widetilde{F}[i]\lambda^i$ of degree at most $\gamma_2 - 1$ such that

$$B_{j+\gamma_2} = \sum_{i=0}^{\gamma_2 - 1} B_{i+j}\widetilde{F}[i], \qquad \forall j \geq 0.$$

This means

$$B_{\gamma_2 + j} = \begin{bmatrix} B_j & \cdots & B_{j+\gamma_2-1} \end{bmatrix} M, \quad \forall j \geq 0,$$

where

$$M = \begin{bmatrix} \widetilde{F}[0] \\ \vdots \\ \widetilde{F}[\gamma_2 - 1] \end{bmatrix} \in \mathbb{F}^{\beta_2\gamma_2 \times \beta_2}.$$

Thus,

$$\begin{bmatrix} B_j & \cdots & B_{\gamma_2+j} \end{bmatrix} = \begin{bmatrix} B_j & \cdots & B_{\gamma_2+j-1} \end{bmatrix} \begin{bmatrix} I & M \end{bmatrix}$$

and

$$\begin{bmatrix} B_i & \cdots & B_{\gamma_2+i+j} \end{bmatrix} = \begin{bmatrix} B_i & \cdots & B_{\gamma_2+i+j-1} \end{bmatrix} \begin{bmatrix} I & 0 & 0 \\ 0 & I & M \end{bmatrix}$$

for every $i \geq 0$ and $j \geq 0$. Thus,

$$H(\nu_1, \nu_2 + 1) = H(\nu_1, \nu_2) \begin{bmatrix} I & 0 & 0 \\ 0 & I & M \end{bmatrix}$$

and $\text{rank}(H(\nu_1, \nu_2 + 1)) \leq \text{rank}(H(\nu_1, \nu_2))$. ■

Lemma 4.5 only gives a bound on the required number of columns in the block Hankel matrix. For the required number of rows, we need a new definition.

Definition 4.7. Let γ_1 be the first index such that the block Hankel matrix

$$H(\gamma_1, \gamma_2 + 1) = \begin{bmatrix} B_0 & B_1 & \cdots & B_{\gamma_2} \\ B_1 & B_2 & \cdots & B_{\gamma_2+1} \\ \vdots & \vdots & \ddots & \vdots \\ B_{\gamma_1-1} & B_{\gamma_1} & \cdots & B_{\gamma_1+\gamma_2-1} \end{bmatrix}$$

has maximal rank. In other words, for any $\nu_1 \geq \gamma_1$,

$$\text{rank}(H(\nu_1, \gamma_2 + 1)) = \text{rank}(H(\gamma_1, \gamma_2 + 1)).$$

We can show that the degree of the minimal polynomial f of the matrix sequence provides an upper bound for γ_1, just as Corollary 4.4 on page 83 shows it gives an upper bound for γ_2.

Theorem 4.4. *Let $\mathbb{F}$ be a field, and let the matrix sequence $\{B_i\}_{i=0}^{\infty} \in \left(\mathbb{F}^{\beta_1 \times \beta_2}\right)^{\mathbb{Z}_{\geq 0}}$ with*

$\beta_1, \beta_2 > 0$ be linearly generated by the polynomial g. Then, $\gamma_1 \leq \deg(g)$.

Proof. The proof is by induction, similar to that of Corollary 4.4 on page 83. The goal is to show that $\operatorname{rank}(H(\nu_1, \gamma_2 + 1)) = \operatorname{rank}(H(\deg(g), \gamma_2 + 1))$ for all $\nu_1 \geq \deg(g)$.

Let $d = \deg(g)$ and $g = \sum_{i=0}^{d} g[i]\lambda^i$. Suppose $\operatorname{rank}(H(\nu_1, \gamma_2+1)) = \operatorname{rank}(H(d, \gamma_2+1))$. Then,

$$\operatorname{rank}(H(\nu_1 + 1, \gamma_2 + 1)) \geq \operatorname{rank}(H(\nu_1, \gamma_2 + 1))$$

because $H(\nu_1, \gamma_2 + 1)$ is a submatrix of $H(\nu_1 + 1, \gamma_2 + 1)$. Also,

$$\sum_{i=0}^{d} g[i]B_{j+i} = 0^{\beta_1 \times \beta_2}, \qquad \forall j \geq 0.$$

since g generates the matrix sequence. Thus,

$$g[d]B_{j+d} = -\sum_{i=0}^{d-1} g[i]B_{j+i}, \qquad \forall j \geq 0,$$

and

$$B_{j+d} = -\frac{1}{g[d]} \sum_{i=0}^{d-1} g[i]B_{j+i}, \qquad \forall j \geq 0.$$

Let M be the block matrix

$$M = -\frac{1}{g[d]} \begin{bmatrix} g[0]I & \cdots & g[d-1]I \end{bmatrix} \in \mathbb{F}^{\beta_1 \times d\beta_1},$$

so

$$B_{j+d} = -\frac{1}{g[d]} \begin{bmatrix} g[0]I & \cdots & g[d-1]I \end{bmatrix} \begin{bmatrix} B_j \\ \vdots \\ B_{j+d-1} \end{bmatrix}, \qquad \forall j \geq 0,$$

and

$$\begin{bmatrix} B_j \\ \vdots \\ B_{j+d} \end{bmatrix} = \begin{bmatrix} I \\ \\ M \end{bmatrix} \begin{bmatrix} B_j \\ \vdots \\ B_{j+d-1} \end{bmatrix}, \qquad \forall j \geq 0,$$

which means

$$\begin{bmatrix} B_i \\ \vdots \\ B_{i+j+d} \end{bmatrix} = \begin{bmatrix} I & 0 \\ 0 & I \\ 0 & M \end{bmatrix} \begin{bmatrix} B_i \\ \vdots \\ B_{i+j+d-1} \end{bmatrix},$$

for all $j \geq 0$ and $i \geq 0$. Thus,

$$H(\nu_1 + 1, \gamma_2 + 1) = \begin{bmatrix} I & 0 \\ 0 & I \\ 0 & M \end{bmatrix} H(\nu_1, \gamma_2 + 1)$$

and $\text{rank}(H(\nu_1 + 1, \gamma_2 + 1)) \leq \text{rank}(H(\nu_1, \gamma_2 + 1))$. ■

Theorem 4.4 says the degree of the minimal generating matrix polynomial is no greater than the degree of the minimal polynomial of the matrix sequence. For example, the γ_1 for Example 4.1 on page 83 is also bounded by $\gamma_1 \leq 2$.

The new quantity γ_1, along with Lemma 4.5 on page 85, gives the rank requirements

needed to decide if a vector polynomial generates the matrix sequence $\{B_i\}_{i=0}^{\infty}$.

Corollary 4.5 (to Lemma 4.5 on page 85). *Let $\mathbb{F}$ be a field, and let the matrix sequence $\{B_i\}_{i=0}^{\infty} \in \left(\mathbb{F}^{\beta_1 \times \beta_2}\right)^{\mathbb{Z}_{\geq 0}}$ with $\beta_1, \beta_2 > 0$ be linearly generated. Let $\nu_1 \geq \gamma_1$ and $\nu_2 \geq \gamma_2$. Then,*

$$\operatorname{rank}(H(\nu_1, \nu_2 + 1)) = \operatorname{rank}(H(\gamma_1, \gamma_2 + 1)).$$

Proof. The proof follows from Definition 4.7 on page 87 and Lemma 4.5 on page 85. ■

This maximality of rank means that γ_1 determines how much of the matrix sequence is required to decide if a vector polynomial generates the matrix sequence.

Theorem 4.5. *Let $\mathbb{F}$ be a field, and let the matrix sequence $\{B_i\}_{i=0}^{\infty} \in \left(\mathbb{F}^{\beta_1 \times \beta_2}\right)^{\mathbb{Z}_{\geq 0}}$ with $\beta_1, \beta_2 > 0$ be linearly generated. Let $\nu_1 \geq \gamma_1$. Then, a vector polynomial*

$$C = \sum_{i=0}^{d} C[i]\lambda^i \in \mathbb{F}^{\beta_2}[\lambda]$$

generates the matrix sequence $\{B_i\}_{i=0}^{\infty}$ from the right if and only if

$$\sum_{j=0}^{d} B_{i+j} C[j] = 0^{\beta_1}, \qquad 0 \leq i \leq \nu_1 - 1. \tag{4.2}$$

Proof. Suppose C generates the matrix sequence from the right. Then,

$$\sum_{j=0}^{d} B_{i+j} C[j] = 0^{\beta_1}, \qquad i \geq 0,$$

and C satisfies Equation (4.2).

Conversely, suppose C satisfies Equation (4.2). Then,

$$\sum_{j=0}^{\nu_2} B_{i+j}C[j] = 0^{\beta_1}, \qquad 0 \le i \le \gamma_1 - 1,$$

where $\nu_2 = \max\{d, \gamma_2\}$ and $C[d+1] = \cdots = C[\nu_2] = 0$ if $d < \nu_2$. This means the coefficients of C form a vector in the right null space of $H(\gamma_1, \nu_2 + 1)$,

$$\begin{bmatrix} B_0 & B_1 & \cdots & B_{\nu_2} \\ B_1 & B_2 & \cdots & B_{\nu_2+1} \\ \vdots & \vdots & \ddots & \vdots \\ B_{\gamma_1-1} & B_{\gamma_1} & \cdots & B_{\gamma_1+\nu_2-1} \end{bmatrix} \begin{bmatrix} C[0] \\ C[1] \\ \vdots \\ C[\nu_2] \end{bmatrix} = 0^{\gamma_1\beta_1}.$$

If C does not generate the matrix sequence, there exists a $k \ge \gamma_1$ such that

$$\sum_{j=0}^{\nu_2} B_{k+j}C[j] \neq 0^{\beta_1},$$

and the vector formed by the coefficients of C is not in the right null space of the larger block Hankel matrix $H(k+1, \nu_2+1)$. However, $\text{rank}(H(k+1, \nu_2+1)) = \text{rank}(H(\gamma_1, \nu_2+1))$ by Corollary 4.5 on the previous page. Thus, the two matrices must have the same right null spaces, and C must generate the matrix sequence. ■

There is no requirement in Theorem 4.5 that $C[d] \neq 0$, so we only know that the degree of C as a vector polynomial is bounded by d, $\deg(C) \le d$.

Because γ_2 is the degree of the minimal generating matrix polynomial F, we can set $d = \gamma_2$. Then, Theorem 4.5 says that if the vector polynomial $C = \sum_{i=0}^{\gamma_2} C[i]\lambda^i$ is a column of the minimal generating matrix polynomial F, its coefficients must form a solution to the $\beta_1\gamma_1 \times \beta_2(\gamma_2+1)$ homogeneous block Hankel system defined by $H(\gamma_1, \gamma_2+1)$,

$$\begin{bmatrix} B_0 & B_1 & \cdots & B_{\gamma_2} \\ B_1 & B_2 & \cdots & B_{\gamma_2+1} \\ \vdots & \vdots & \ddots & \vdots \\ B_{\gamma_1-1} & B_{\gamma_1} & \cdots & B_{\gamma_2+\gamma_1-1} \end{bmatrix} \begin{bmatrix} C[0] \\ C[1] \\ \vdots \\ C[\gamma_2] \end{bmatrix} = 0^{\gamma_1\beta_1},$$

or, equivalently, the $\beta_1\gamma_1 \times \beta_2(\gamma_2 + 1)$ homogeneous block Toeplitz system,

$$\begin{bmatrix} B_{\gamma_2} & B_{\gamma_2-1} & \cdots & B_0 \\ B_{\gamma_2+1} & B_{\gamma_2} & \cdots & B_1 \\ \vdots & \vdots & \ddots & \vdots \\ B_{\gamma_2+\gamma_1-1} & B_{\gamma_2+\gamma_1-2} & \cdots & B_{\gamma_1-1} \end{bmatrix} \begin{bmatrix} C[\gamma_2] \\ C[\gamma_2 - 1] \\ \vdots \\ C[0] \end{bmatrix} = 0^{\gamma_1\beta_1}.$$

Finding a basis for the solutions to these homogeneous systems over $\mathbb{F}$ is one way to find a basis over $\mathbb{F}[\lambda]$ for the generating vector polynomials of the linearly generated matrix sequence $\{B_i\}_{i=0}^{\infty}$ and, thus, the minimal generating matrix polynomial F.

The reversal of a polynomial g with respect to the degree d for $d \geq \deg(g)$ is $\mathrm{rev}_d(g) = \lambda^d g(1/\lambda)$ (von zur Gathen and Gerhard, 1999, page 244). Similarly, the reversal of a vector polynomial C with respect to the degree d for $d \geq \deg(C)$ is $\mathrm{rev}_d(C) = \lambda^d C(1/\lambda)$.

The degree of the reversal is at most d, $\deg(\hat{C}) \leq d$. Then, Theorem 4.5 on page 90 gives a block version of the modular equivalence solved by the Berlekamp-Massey algorithm.

Theorem 4.6. *Let $\mathbb{F}$ be a field, and let the matrix sequence $\{B_i\}_{i=0}^{\infty} \in \left(\mathbb{F}^{\beta_1 \times \beta_2}\right)^{\mathbb{Z}_{\geq 0}}$ with $\beta_1, \beta_2 > 0$ be linearly generated. Let*

$$C = \sum_{i=0}^{d} C[i]\lambda^i \in \mathbb{F}^{\beta_2}\left[\lambda\right],$$

$\nu_1 \geq \gamma_1$, and $\nu_2 \geq d$. Then, C is a right generating vector polynomial for the matrix sequence if and only if its vector polynomial reversal $\hat{C}$ with respect to degree d satisfies the equivalence relation

$$\left(\sum_{i=0}^{\nu_1+\nu_2-1} B_i\lambda^i\right)\hat{C}(\lambda) \equiv C^{(res)}(\lambda) \pmod{\lambda^{\nu_1+\nu_2}} \tag{4.3}$$

for a vector polynomial $C^{(res)}$ of degree at most $d-1$.

Proof. The vector polynomial $\hat{C}$ is

$$\hat{C} = \mathrm{rev}_d(C) = \lambda^d C\left(\frac{1}{\lambda}\right) = \sum_{i=0}^{d} C[d-i]\lambda^i.$$

For any vector polynomial C,

$$\begin{aligned}\left(\sum_{i=0}^{\infty} B_i\lambda^i\right)\hat{C} &= \sum_{i=0}^{d-1}\left(\sum_{j=d-i}^{d} B_{j+i-d}C[j]\right)\lambda^i + \sum_{i=d}^{\infty}\left(\sum_{j=0}^{d} B_{j+i-d}C[j]\right)\lambda^i \\ &= \sum_{i=0}^{d-1}\left(\sum_{j=d-i}^{d} B_{j+i-d}C[j]\right)\lambda^i + \sum_{i=0}^{\infty}\left(\sum_{j=0}^{d} B_{j+i}C[j]\right)\lambda^{i+d}.\end{aligned}$$

If C generates the matrix sequence from the right,

$$\sum_{j=0}^{d} B_{i+j}C[j] = 0^{\beta_1}, \qquad 0 \le i \le \nu_1 + \nu_2 - d - 1,$$

by Theorem 4.5 on page 90. Let $C^{(\mathrm{res})}$ be the vector polynomial

$$C^{(\mathrm{res})} = \sum_{i=0}^{d-1} \left(\sum_{j=d-i}^{d} B_{j+i-d}C[j] \right) \lambda^i,$$

which has degree at most $d-1$. Then,

$$\begin{aligned}
\left(\sum_{i=0}^{\infty} B_i \lambda^i \right) \hat{C} - C^{(\mathrm{res})} &= \sum_{i=0}^{\infty} \left(\sum_{j=0}^{d} B_{j+i}C[j] \right) \lambda^{i+d} \\
&= \sum_{i=\gamma_1+\gamma_2-d}^{\infty} \left(\sum_{j=0}^{d} B_{j+i}C[j] \right) \lambda^{i+d} \\
&= \sum_{i=\gamma_1+\gamma_2}^{\infty} \left(\sum_{j=0}^{d} B_{j+i-d}C[j] \right) \lambda^{i}
\end{aligned}$$

and the equivalence relation (4.3) holds.

If, on the other hand, the equivalence relation (4.3) is true, $C^{(\mathrm{res})}$ must be the vector polynomial

$$C^{(\mathrm{res})} = \sum_{i=0}^{d-1} \left(\sum_{j=d-i}^{d} B_{j+i-d}C[j] \right) \lambda^i$$

because of the degree bound $\deg(C^{(\mathrm{res})}) \le d-1$. Thus,

$$\sum_{i=d}^{\nu_1+\nu_2-1} \left(\sum_{j=0}^{d} B_{j+i-d}C[j] \right) \lambda^i = 0$$

and

$$\sum_{j=0}^{d} B_{j+i-d} C[j] = 0, \qquad d \leq i \leq \nu_1 + \nu_2 - 1.$$

In other words,

$$\sum_{j=0}^{d} B_{i+j} C[j] = 0, \qquad 0 \leq i \leq \nu_1 + \nu_2 - d - 1,$$

and C generates the matrix sequence by Theorem 4.5 on page 90. ■

Again, Theorem 4.6 only requires the degree of C be no more than d, $\deg(C) \leq d$.

The degree of the minimal generating matrix polynomial F of the matrix sequence $\{B_i\}_{i=0}^{\infty}$ is at most γ_2 by Definition 4.6 on page 83. This means the columns of F have degrees at most γ_2, and we have the following requirement for the columns of F.

Corollary 4.6. *Let $\mathbb{F}$ be a field, and let the matrix sequence $\{B_i\}_{i=0}^{\infty} \in \left(\mathbb{F}^{\beta_1 \times \beta_2}\right)^{\mathbb{Z}_{\geq 0}}$ with $\beta_1, \beta_2 > 0$ be linearly generated. Let $\nu_1 \geq \gamma_1$ and $\nu_2 \geq \gamma_2$. If*

$$C = \sum_{i=0}^{d} C[i] \lambda^i \in \mathbb{F}^{\beta_2}[\lambda]$$

is a column of the minimal generating matrix polynomial F for the matrix sequence $\{B_i\}_{i=0}^{\infty}$, then the vector polynomial reversal $\hat{C}$ of C with respect to degree d must satisfy the equivalence relation

$$\left(\sum_{i=0}^{\nu_1+\nu_2-1} B_i \lambda^i \right) \hat{C}(\lambda) \equiv C^{(res)}(\lambda) \pmod{\lambda^{\nu_1+\nu_2}}$$

for a vector polynomial $C^{(res)}$ of degree at most $d-1$.

Proof. The proof follows from Theorem 4.6 on page 93 and Definition 4.6 on page 83. ■

As in Theorem 4.5 on page 90 and Theorem 4.6 on page 93, Corollary 4.6 requires only that $d \geq \deg(C)$. These results are all useful in computing the minimal generating matrix polynomial F. The rank of the block Hankel matrix $H(\nu_1, \nu_2 + 1)$ also gives a lower bound on the determinantal degree of F that will be useful in the block Wiedemann method.

Lemma 4.6. *Let $\mathbb{F}$ be a field, and let the matrix sequence $\{B_i\}_{i=0}^{\infty} \in \left(\mathbb{F}^{\beta_1 \times \beta_2}\right)^{\mathbb{Z}_{\geq 0}}$ with $\beta_1, \beta_2 > 0$ be linearly generated. Then, for any $\nu_1 \geq 1$ and $\nu_2 \geq 0$, the rank of the block Hankel matrix*

$$H(\nu_1, \nu_2 + 1) = \begin{bmatrix} B_0 & B_1 & \cdots & B_{\nu_2} \\ B_1 & B_2 & \cdots & B_{\nu_2+1} \\ \vdots & \vdots & \ddots & \vdots \\ B_{\nu_1-1} & B_{\nu_1} & \cdots & B_{\nu_1+\nu_2-1} \end{bmatrix}$$

is at most the determinantal degree of the minimal generating matrix polynomial F of the matrix sequence $\{B_i\}_{i=0}^{\infty}$,

$$\operatorname{rank}(H(\nu_1, \nu_2 + 1)) \leq \deg(\det(F)).$$

Proof. The proof is accomplished by examining the right null space of the block Hankel

matrix $H(\nu_1, \nu_2 + 1)$. Let

$$C_j = \sum_{i=0}^{d_j} C_j[i]\lambda^i \in \mathbb{F}^{\beta_2}[\lambda]$$

be the j-th column of the minimal generating matrix polynomial F and $d_j = \deg(C_j)$. Then, by construction, $C_1, \ldots, C_{\beta_2}$ are linearly independent over $\mathbb{F}[\lambda]$ and the sum of their degrees is the determinantal degree of F,

$$\deg(\det(F)) = \sum_{j=1}^{\beta_2} \deg(C_j) = \sum_{j=1}^{\beta_2} d_j.$$

Because each of the C_j generate the matrix sequence from the right, if $d_j \leq \nu_2$, the $\nu_2 - d_j + 1$ vectors

$$\begin{aligned}
\vec{C}_{j,0} &= \begin{bmatrix} C_j[0] & C_j[1] & \cdots & C_j[d_j] & 0 & 0 & \cdots & 0 \end{bmatrix}^{\mathrm{T}}, \\
\vec{C}_{j,1} &= \begin{bmatrix} 0 & C_j[0] & C_j[1] & \cdots & C_j[d_j] & 0 & \cdots & 0 \end{bmatrix}^{\mathrm{T}}, \\
&\vdots \\
\vec{C}_{j,\nu_2-d_j} &= \begin{bmatrix} 0 & 0 & \cdots & 0 & C_j[0] & C_j[1] & \cdots & C_j[d_j] \end{bmatrix}^{\mathrm{T}}
\end{aligned}$$

are in the right null space of $H(\nu_1, \nu_2 + 1)$. If $d_j > \nu_2$, the column C_j does not contribute any vectors to the right null space of $H(\nu_1, \nu_2 + 1)$. Because $C_1, \ldots, C_{\beta_2}$ are linearly independent over $\mathbb{F}[\lambda]$, the set of vectors

$$\{\vec{C}_{j,i} \mid 1 \leq j \leq \beta_2, 0 \leq i \leq \nu_2 - d_j\} \subset \mathrm{Null}(H(\nu_1, \nu_2 + 1)) \subset \mathbb{F}^{(\gamma_2+1)\beta_2}$$

is linearly independent over $\mathbb{F}$. Thus,

$$\begin{aligned}\dim(\text{Null}(H(\nu_1, \nu_2+1))) &\geq \sum_{\substack{1\leq j\leq \beta_2 \\ d_j \leq \nu_2}} (\nu_2 - d_j + 1) \geq \sum_{j=1}^{\beta_2} (\nu_2 - d_j + 1) \\ &= \beta_2(\nu_2+1) - \sum_{j=1}^{\beta_2} d_j\end{aligned}$$

and the rank of $H(\nu_1, \nu_2+1)$ is

$$\begin{aligned}\text{rank}(H(\nu_1, \nu_2+1)) &= \beta_2(\nu_2+1) - \dim(\text{Null}(H(\nu_1, \nu_2+1))) \\ &\leq \sum_{j=1}^{\beta_2} d_j = \deg(\det(F)).\end{aligned}$$

■

We will use Lemma 4.6 when we discuss the block Wiedemann method in Section 4.3, but first we turn our attention to computing the minimal generating matrix polynomial F for any linearly generated matrix sequence $\{B_i\}_{i=0}^{\infty}$.

4.2 Fast Power Hermite-Padé Solver

In this section, we examine how to compute the minimal generating matrix polynomial F of any linearly generated matrix sequence $\{B_i\}_{i=0}^{\infty}$ using the Beckermann-Labahn Fast Power Hermite-Padé Solver (FPHPS) algorithm. We begin by examining the relationship between the power Hermite-Padé approximation problem, the vector Hermite-Padé approximation problem, and computing generating vector polynomials of the matrix sequence. Then, we investigate properties of the particular power Hermite-Padé approx-

imation problem to be solved in order to show it provides a basis over $\mathbb{F}[\lambda]$ for the right generating vector polynomials. The minimal generating matrix polynomial can then be constructed by computing the Popov form of the matrix whose columns are these basis elements.

Definition 4.8. The polynomial tuple $\mathscr{P} = (\mathscr{P}^{[1]}, \ldots, \mathscr{P}^{[m]})$ is a vector Hermite-Padé approximant for the vector polynomials $\mathscr{G}^{[1]}, \ldots, \mathscr{G}^{[m]} \in \mathbb{F}^s[\lambda]$ of type $(\mathscr{N}, \sigma)$ with the multiindex $\mathscr{N} = (\mathscr{N}^{[1]}, \ldots, \mathscr{N}^{[m]})$ if there exists a vector polynomial $R \in \mathbb{F}^s[\lambda]$ such that

$$\sum_{i=1}^{m} \mathscr{G}^{[i]}(\lambda)\mathscr{P}^{[i]}(\lambda) = \lambda^{\sigma} R(\lambda)$$

and $\deg(\mathscr{P}^{[i]}) \leq \mathscr{N}^{[i]}$ for $1 \leq i \leq m$ (Beckermann and Labahn, 1994, Example 2.5).

Beckermann and Labahn (1994) introduce a generalized Hermite-Padé approximation problem, called a power Hermite-Padé approximation problem.

Definition 4.9 (Beckermann and Labahn, 1994, Definition 1.1). The polynomial tuple $\mathscr{P} = (\mathscr{P}^{[1]}, \ldots, \mathscr{P}^{[m]})$ is a power Hermite-Padé approximant of the type $(\mathscr{N}, \sigma, s)$ given by the polynomial tuple $\mathscr{F} = (\mathscr{F}^{[1]}, \ldots, \mathscr{F}^{[m]})$ with the multiindex $\mathscr{N} = (\mathscr{N}^{[1]}, \ldots, \mathscr{N}^{[m]})$ if there exists a polynomial $R' \in \mathbb{F}[\lambda]$ such that

$$\mathscr{F}(\lambda) \cdot \mathscr{P}(\lambda^s) = \sum_{i=1}^{m} \mathscr{F}^{[i]}(\lambda)\mathscr{P}^{[i]}(\lambda^s) = \lambda^{\sigma} R'(\lambda)$$

and $\deg(\mathscr{P}^{[i]}) \leq \mathscr{N}^{[i]}$ for $1 \leq i \leq m$.

We first look at the equivalence of the vector Hermite-Padé approximation problem and the power Hermite-Padé approximation problem.

Lemma 4.7. *Let $\mathbb{F}$ be a field, $\mathscr{G}^{[1]}, \ldots, \mathscr{G}^{[m]} \in \mathbb{F}^s[\lambda]$, and $\mathscr{F} = (\mathscr{F}^{[1]}, \ldots, \mathscr{F}^{[m]})$ be the polynomial tuple given by $\mathscr{F}^{[i]}(\lambda) = (1, \lambda, \lambda^2, \ldots, \lambda^{s-1}) \cdot \mathscr{G}^{[i]}(\lambda^s)$. Then, a polynomial tuple $\mathscr{P} = (\mathscr{P}^{[1]}, \ldots, \mathscr{P}^{[m]})$ solves the vector Hermite-Padé approximation problem*

$$\sum_{i=1}^{m} \mathscr{G}^{[i]}(\lambda)\mathscr{P}^{[i]}(\lambda) = \lambda^{\sigma} R(\lambda) \in \mathbb{F}^s[\lambda] \tag{4.4}$$

of type $(\mathscr{N}, \sigma)$ if and only if it solves the power Hermite-Padé approximation problem

$$\mathscr{F}(\lambda) \cdot \mathscr{P}(\lambda^s) = \sum_{i=1}^{m} \mathscr{F}^{[i]}(\lambda)\mathscr{P}^{[i]}(\lambda^s) = \lambda^{\sigma s} R'(\lambda) \in \mathbb{F}[\lambda] \tag{4.5}$$

of type $(\mathscr{N}, \sigma s, s)$.

Proof. $\mathscr{P}$ solves the vector Hermite-Padé approximation problem (4.4) if and only if

$$\sum_{i=1}^{m} \mathscr{G}^{[i,j]}(\lambda)\mathscr{P}^{[i]}(\lambda) = \lambda^{\sigma} R(\lambda), \qquad 1 \le j \le m,$$

where

$$\mathscr{G}^{[i]} = \begin{bmatrix} \mathscr{G}^{[i,1]} & \mathscr{G}^{[i,2]} & \ldots & \mathscr{G}^{[i,s]} \end{bmatrix}^{\mathrm{T}} \in \mathbb{F}^s[\lambda]$$

and

$$R = \begin{bmatrix} R^{[1]} & R^{[2]} & \ldots & R^{[s]} \end{bmatrix}^{\mathrm{T}} \in \mathbb{F}^s[\lambda],$$

which is equivalent to

$$\sum_{i=1}^{m} \mathscr{G}^{[i,j]}(\lambda^s)\mathscr{P}^{[i]}(\lambda^s) = \lambda^{\sigma s} R^{[j]}(\lambda^s), \qquad 1 \le j \le s,$$

and

$$\sum_{i=1}^{m} \lambda^{j-1}\mathscr{G}^{[i,j]}(\lambda^s)\mathscr{P}^{[i]}(\lambda^s) = \lambda^{\sigma s+j-1} R^{[j]}(\lambda^s), \qquad 1 \le j \le s.$$

Thus, $\mathscr{P}$ solves the vector Hermite-Padé approximation problem (4.4) if and only if

$$\sum_{j=1}^{s}\sum_{i=1}^{m} \lambda^{j-1}\mathscr{G}^{[i,j]}(\lambda^s)\mathscr{P}^{[i]}(\lambda^s) = \sum_{j=1}^{s} \lambda^{\sigma s+j-1} R^{[j]}(\lambda^s)$$

or, equivalently,

$$\sum_{i=1}^{m} \left(\sum_{j=1}^{s} \lambda^{j-1}\mathscr{G}^{[i,j]}(\lambda^s) \right) \mathscr{P}^{[i]}(\lambda^s) = \lambda^{\sigma s} \sum_{j=1}^{s} \lambda^{j-1} R^{[j]}(\lambda^s).$$

Therefore, the vector Hermite-Padé approximation problem (4.4) is equivalent to the power Hermite-Padé approximation problem (4.5) where $R'(\lambda) = \sum_{j=1}^{s} \lambda^{j-1} R^{[j]}(\lambda^s)$. ■

Lemma 4.7 gives a relationship between the vector Hermite-Padé approximation problem and the power Hermite-Padé approximation problem. The power substitution $\lambda \leftarrow \lambda^s$ and the vector $(1, \lambda, \lambda^2, \ldots, \lambda^{s-1})$ in the computation of $\mathscr{F}$ enforce a unique relationship between the rows of the vector Hermite-Padé approximant (4.4) and the powers of the power Hermite-Padé approximant (4.5). Namely, the coefficient of the λ^i term in the j-th row of the vector Hermite-Padé approximant is the coefficient of the λ^{is+j} term in

the power Hermite-Padé approximant.

The modular equivalence of Theorem 4.6 on page 93 shows the reversals of a generating vector polynomial must solve the power Hermite-Padé approximation problem in much the same way that the reversals of a generating polynomial are Padé approximations (von zur Gathen and Gerhard, 1999, Lemma 12.8).

Theorem 4.7. *Let $\mathbb{F}$ be a field, and let the matrix sequence $\{B_i\}_{i=0}^{\infty} \in \left(\mathbb{F}^{\beta_1 \times \beta_2}\right)^{\mathbb{Z}_{\geq 0}}$ with $\beta_1, \beta_2 > 0$ be linearly generated. Let*

$$C = \sum_{i=0}^{d} C[i]\lambda^i \in \mathbb{F}^{\beta_2}[\lambda],$$

$\nu_1 \geq \gamma_1$, $\nu_2 \geq d$, and $\sigma = (\nu_1 + \nu_2)\beta_1$. Then, C is a right generating vector polynomial for the matrix sequence $\{B_i\}_{i=0}^{\infty}$ if and only if the vector polynomial reversal $\hat{C}$ of C with respect to degree d and a vector polynomial $C^{(res)}$ of degree at most $d-1$ form a solution

$$\mathscr{P} = \begin{bmatrix} C^{(res)} \\ \\ \hat{C} \end{bmatrix} \in \mathbb{F}^{\beta_1+\beta_2}[\lambda]$$

to the power Hermite-Padé approximation problem of type $(\mathscr{N}, \sigma, \beta_1)$ given by the multi-index $\mathscr{N} = (\mathscr{N}^{[1]}, \ldots, \mathscr{N}^{[\beta_1+\beta_2]})$ with

$$\mathscr{N}^{[i]} = \begin{cases} \nu_2 - 1 & \text{if } 1 \leq i \leq \beta_1, \\ \\ \nu_2 & \text{if } \beta_1 + 1 \leq i \leq \beta_1 + \beta_2 \end{cases}$$

and the polynomial tuple $\mathscr{F} = (\mathscr{F}^{[1]}, \ldots, \mathscr{F}^{[\beta_1+\beta_2]})$ *with*

$$\mathscr{F}^{[i]}(\lambda) = (1, \lambda, \lambda^2, \ldots, \lambda^{\beta_1-1}) \cdot \mathscr{G}^{[i]}(\lambda^{\beta_1})$$

where $\mathscr{G}^{[i]}$ *is the* i*-th column of the matrix polynomial*

$$\mathscr{G}(\lambda) = \left[I \quad -\left(\sum_{i=0}^{\nu_1+\nu_2-1} B_i \lambda^i\right) \right] \in \mathbb{F}^{\beta_1 \times (\beta_1+\beta_2)}[\lambda].$$

Proof. By Theorem 4.6 on page 93, C generates the matrix sequence from the right if and only if there exists some $R(\lambda) \in \mathbb{F}^{\beta_1}[\lambda]$ such that

$$C^{(\mathrm{res})}(\lambda) - \left(\sum_{i=0}^{\nu_1+\nu_2-1} B_i \lambda^i \right) \hat{C}(\lambda) = \lambda^{\nu_1+\nu_2} R(\lambda)$$

or

$$\left[I \quad -\left(\sum_{i=0}^{\nu_1+\nu_2-1} B_i \lambda^i\right) \right] \begin{bmatrix} C^{(\mathrm{res})}(\lambda) \\ \hat{C}(\lambda) \end{bmatrix} = \lambda^{\nu_1+\nu_2} R(\lambda)$$

for some vector polynomial $C^{(\mathrm{res})}$ of degree at most $d-1$. This is equivalent to the vector Hermite-Padé approximation problem

$$\sum_{i=1}^{\beta_1+\beta_2} \mathscr{G}^{[i]}(\lambda) \mathscr{P}^{[i]}(\lambda) = \lambda^{\nu_1+\nu_2} R(\lambda)$$

of type $(\mathscr{N}, \nu_1+\nu_2)$, and is, thus, equivalent to the desired power Hermite-Padé approximation problem by Lemma 4.7 on page 100. ∎

Theorem 4.7 shows a generating vector polynomial is related to a solution of a corresponding power Hermite-Padé approximation problem.

For example, consider the matrix sequence described in Example 4.1 on page 83. If $\nu_1 = \gamma_1 = 2$ and $\nu_2 = \gamma_2 = 2$, the corresponding power Hermite-Padé approximation problem is given by the the matrix polynomials

$$\sum_{i=0}^{\nu_1+\nu_2-1} B_i\lambda^i = \begin{bmatrix} \lambda + \lambda^2 + 2\lambda^3 & 1 + \lambda^2 + \lambda^3 \\ 1 + 3\lambda + 4\lambda^2 + 7\lambda^3 & 2 + \lambda + 3\lambda^2 + 4\lambda^3 \end{bmatrix}$$

and

$$\begin{aligned} \mathscr{G}(\lambda) &= \left[I \quad -\left(\textstyle\sum_{i=0}^{\nu_1+\nu_2-1} B_i\lambda^i\right) \right] \\ &= \begin{bmatrix} 1 & 0 & -\lambda - \lambda^2 - 2\lambda^3 & -1 - \lambda^2 - \lambda^3 \\ 0 & 1 & -1 - 3\lambda - 4\lambda^2 - 7\lambda^3 & -2 - \lambda - 3\lambda^2 - 4\lambda^3 \end{bmatrix}. \end{aligned}$$

Thus, because the columns of the minimum generating matrix polynomial F have degree at most $\gamma_2 = 2$, they are related to the solutions of the power Hermite-Padé approximation problem given by $\sigma = (\nu_1 + \nu_2)\beta_1 = 8$, the polynomial tuple

$$\mathscr{F} = \begin{bmatrix} 1 \\ \lambda \\ -\lambda - \lambda^2 - 3\lambda^3 - \lambda^4 - 4\lambda^5 - 2\lambda^6 - 7\lambda^7 \\ -1 - 2\lambda - \lambda^3 - \lambda^4 - 4\lambda^5 - \lambda^6 - 4\lambda^7 \end{bmatrix}^{\mathrm{T}},$$

and the multiindex $\mathscr{N} = (1, 1, 2, 2)$. However, we do not know that any solution to this power Hermite-Padé approximation problem is related to a generating vector polynomial. For that, we need another definition.

Definition 4.10 (Beckermann and Labahn, 1994, Definition 3.1). Let the polynomial tuple $\mathscr{P} = (\mathscr{P}^{[1]}, \ldots, \mathscr{P}^{[m]})$ be a solution to a power Hermite-Padé approximation problem of type $(\mathscr{N}, \sigma, s)$. Then, the defect of $\mathscr{P}$ is one more than the minimum difference in degree of a member polynomial and the corresponding degree bound,

$$\operatorname{dct}(\mathscr{P}) = \min_{1 \le i \le \beta_1 + \beta_2} \left\{ \mathscr{N}^{[i]} + 1 - \deg(\mathscr{P}^{[i]}) \right\}.$$

The defect of a polynomial tuple is one minus its τ-degree in Definition 4 of (Van Barel and Bultheel, 1991), $\operatorname{dct}(\mathscr{P}) = 1 - \tau\text{-deg}(\mathscr{P})$. The following theorem shows that given a power Hermite-Padé approximation problem corresponding to a linearly generated matrix sequence $\{B_i\}_{i=0}^{\infty}$ and a bound ν_2 on generating vector polynomial degrees, the defect $\operatorname{dct}(\mathscr{P})$, and thus the τ-degree, of the polynomial tuple solution $\mathscr{P}$ to the power Hermite-Padé approximation problem relates the polynomial tuple to a generating vector polynomial of the linearly generated matrix sequence through a vector polynomial reversal of degree $\nu_2 + 1 - \operatorname{dct}(\mathscr{P})$.

Theorem 4.8. *Let $\mathbb{F}$ be a field, and let the matrix sequence $\{B_i\}_{i=0}^{\infty} \in \left(\mathbb{F}^{\beta_1 \times \beta_2}\right)^{\mathbb{Z}_{\ge 0}}$ with $\beta_1, \beta_2 > 0$ be linearly generated. Let $\nu_1 \ge \gamma_1$, $\nu_2 \ge \gamma_2$, and $\sigma = (\nu_1 + \nu_2)\beta_1$. Let the polynomial tuple $\mathscr{P} = (\mathscr{P}^{[1]}, \ldots, \mathscr{P}^{[\beta_1+\beta_2]})$ be a solution to the power Hermite-Padé approx-*

imation problem of type $(\mathscr{N}, \sigma, \beta_1)$ *given by the multiindex* $\mathscr{N} = (\mathscr{N}^{[1]}, \ldots, \mathscr{N}^{[\beta_1+\beta_2]})$ *with*

$$\mathscr{N}^{[i]} = \begin{cases} \nu_2 - 1 & \text{if } 1 \le i \le \beta_1, \\ \nu_2 & \text{if } \beta_1 + 1 \le i \le \beta_1 + \beta_2 \end{cases}$$

and the polynomial tuple $\mathscr{F} = (\mathscr{F}^{[1]}, \ldots, \mathscr{F}^{[\beta_1+\beta_2]})$ *with*

$$\mathscr{F}^{[i]}(\lambda) = (1, \lambda, \lambda^2, \ldots, \lambda^{\beta_1-1}) \cdot \mathscr{G}^{[i]}(\lambda^{\beta_1})$$

where $\mathscr{G}^{[i]}$ *is the* i*-th column of the matrix polynomial*

$$\mathscr{G}(\lambda) = \left[I \quad -\left(\sum_{i=0}^{\nu_1+\nu_2-1} B_i \lambda^i\right) \right] \in \mathbb{F}^{\beta_1 \times (\beta_1+\beta_2)}[\lambda].$$

Then, the reversal of the vector polynomial

$$\left[\mathscr{P}^{[\beta_1+1]} \quad \mathscr{P}^{[\beta_1+2]} \quad \ldots \quad \mathscr{P}^{[\beta_1+\beta_2]} \right]^T \in \mathbb{F}^{\beta_2}[\lambda]$$

with respect to degree $\nu_2 + 1 - \operatorname{dct}(\mathscr{P})$ *is a right generating vector polynomial for the matrix sequence* $\{B_i\}_{i=0}^{\infty}$.

Proof. Let $d = \nu_2 + 1 - \operatorname{dct}(\mathscr{P})$ and let $C^{(\mathrm{res})}$ and $\hat{C}$ be the vector polynomials

$$C^{(\mathrm{res})} = \left[\mathscr{P}^{[1]} \quad \mathscr{P}^{[2]} \quad \ldots \quad \mathscr{P}^{[\beta_1]} \right]^{\mathrm{T}} \in \mathbb{F}^{\beta_1}[\lambda]$$

and

$$\hat{C} = \left[\mathscr{P}^{[\beta_1+1]} \quad \mathscr{P}^{[\beta_1+2]} \quad \dots \quad \mathscr{P}^{[\beta_1+\beta_2]} \right]^{\mathrm{T}} \in \mathbb{F}^{\beta_2}[\lambda].$$

By Definition 4.10 on page 105, $\deg(C^{(\mathrm{res})}) \leq d-1$ and $\deg(\hat{C}) \leq d$, so there exist $C[1], \dots, C[d] \in \mathbb{F}$ such that $\hat{C} = \sum_{i=0}^{d} C[d-i]\lambda^i$. Thus, $\hat{C}$ is the reversal of the vector polynomial $C = \sum_{i=0}^{d} C[i]\lambda^i$ with respect to degree d that must generate the matrix sequence by Theorem 4.7 on page 102. ■

The Fast Power Hermite-Padé Solver (FPHPS) algorithm (Beckermann and Labahn, 1994, Section 3) computes a basis for the polynomials tuples $\mathscr{P}$ satisfying the power Hermite-Padé approximation problem of type $(\mathscr{N}, \sigma, s)$ and, thus, also for the vector Hermite-Padé approximation problem by Lemma 4.7 on page 100. It does this by iterating through the powers σ of λ so that at the end of the k-th step, the algorithm has computed polynomial tuples $\mathscr{P}_{1,k}, \dots, \mathscr{P}_{m,k}$ and their defects $\mathrm{dct}(\mathscr{P}_{1,k}), \dots, \mathrm{dct}(\mathscr{P}_{m,k})$. After the k-step, any solution $\mathscr{P}$ to the power Hermite-Padé approximation $\mathscr{F}(\lambda) \cdot \mathscr{P}(\lambda^s) \equiv 0 \pmod{\lambda^k}$ can be written as exactly one linear combination of the polynomial tuples $\mathscr{P}_{1,k}, \dots, \mathscr{P}_{m,k}$,

$$\mathscr{P} = g_1(\lambda)\mathscr{P}_{1,k} + \cdots + g_m(\lambda)\mathscr{P}_{m,k},$$

where $\deg(g_j) < \mathrm{dct}(\mathscr{P}_j)$, $1 \leq j \leq m$ (Beckermann and Labahn, 1994, Theorem 3.4). Only the $\mathscr{P}_j$ with positive defect contribute to this basis for the solutions. After $k\beta_1$ steps of the FPHPS algorithm using $s = \beta_1$, $m = \beta_1 + \beta_2$, and $\mathscr{F}$ and $\mathscr{N}$ as in Theorem 4.7 on page 102, the polynomial tuples $\mathscr{P}_{1,k\beta_1}, \dots, \mathscr{P}_{\beta_1+\beta_2,k\beta_1}$ form a basis over $\mathbb{F}[\lambda]$ for the

solutions to the equation

$$\left[I \quad -\left(\sum_{i=0}^{\nu_1+\nu_2-1} B_i \lambda^i\right) \right] \mathscr{P}(\lambda) = \lambda^k R(\lambda).$$

At the end of the $\sigma = (\gamma_1 + \gamma_2)\beta_1$ steps, the output of the algorithm FPHPS is the polynomial tuples $\mathscr{P}_{1,\sigma}, \ldots, \mathscr{P}_{\beta_1+\beta_2,\sigma}$ and their defects $\operatorname{dct}(\mathscr{P}_{1,\sigma}), \ldots, \operatorname{dct}(\mathscr{P}_{\beta_1+\beta_2,\sigma})$.

The FPHPS algorithm presented by Beckermann and Labahn (1994, Section 3) has some freedom in the choice of the index π of the polynomial tuple used to update the approximants. For our purposes, we will consider an implementation that chooses π to have the smallest possible value whenever there is a choice in its value. (See Appendix A.)

We want to show the FPHPS algorithm provides a basis for the generating vector polynomials of the associated block sequence. The first step is to show the algorithm provides the correct number of solutions. To do this, we need to examine the relationship between a solution to the power Hermite-Padé approximation problem and the part of the solution corresponding to the reversal of the generating vector polynomial.

The first result we require is that if the part corresponding to the vector polynomial reversal has a zero constant term, then the entire solution does also. This will be used later to determine the number of solutions that are possible.

Lemma 4.8. *Let $\mathbb{F}$ be a field, $M \in \mathbb{F}^{\beta_1 \times \beta_2}[\lambda]$, $\nu_2 > 0$, and $\sigma \geq \beta_1$. Let the polynomial tuple $\mathscr{P} = (\mathscr{P}^{[1]}, \ldots, \mathscr{P}^{[\beta_1+\beta_2]})$ be a solution to the power Hermite-Padé approximation*

problem of type $(\mathscr{N}, \sigma, \beta_1)$ *given by the polynomial tuple* $\mathscr{F} = (\mathscr{F}^{[1]}, \ldots, \mathscr{F}^{[\beta_1+\beta_2]})$ *with*

$$\mathscr{F}^{[i]}(\lambda) = (1, \lambda, \lambda^2, \ldots, \lambda^{\beta_1-1}) \cdot \mathscr{G}^{[i]}(\lambda^{\beta_1})$$

where $\mathscr{G}^{[i]}$ *is the* i*-th column of the matrix polynomial*

$$\mathscr{G}(\lambda) = \begin{bmatrix} I & M \end{bmatrix} \in \mathbb{F}^{\beta_1 \times (\beta_1+\beta_2)}[\lambda].$$

If $(\mathscr{P}^{[\beta_1+1]}(0), \ldots, \mathscr{P}^{[\beta_1+\beta_2]}(0)) = 0^{\beta_2}$*, then* $(\mathscr{P}^{[1]}(0), \ldots, \mathscr{P}^{[\beta_1]}(0)) = 0^{\beta_1}$*.*

Proof. There exists a polynomial $R(\lambda) \in \mathbb{F}[\lambda]$ such that

$$\mathscr{F}(\lambda) \cdot \mathscr{P}(\lambda^{\beta_1}) = \sum_{i=1}^{\beta_1+\beta_2} \mathscr{F}^{[i]}(\lambda)\mathscr{P}^{[i]}(\lambda^{\beta_1}) = \lambda^{\sigma} R(\lambda) = \lambda^{\beta_1}\left(\lambda^{\sigma-\beta_1} R(\lambda)\right),$$

and, by Lemma 4.7 on page 100, there exists a vector polynomial $R'(\lambda) \in \mathbb{F}^{\beta_1}[\lambda]$ such that

$$\sum_{i=1}^{\beta_1+\beta_2} \mathscr{G}^{[i]}(\lambda)\mathscr{P}^{[i]}(\lambda) = \lambda R'(\lambda).$$

Let $\mathscr{A} = (\mathscr{A}^{[1]}, \ldots, \mathscr{A}^{[\beta_1]})$ and $\mathscr{B} = (\mathscr{B}^{[1]}, \ldots, \mathscr{B}^{[\beta_2]})$ be the polynomial sub-tuples such that $\mathscr{A}^{[i]} = \mathscr{P}^{[i]}$ for $1 \le i \le \beta_1$ and $\mathscr{B}^{[i]} = \mathscr{P}^{[i+\beta_1]}$ for $1 \le i \le \beta_2$. Then,

$$\mathscr{A} + M\mathscr{B} = \lambda R'(\lambda)$$

which means $\mathscr{A}(0) + M(0)\mathscr{B}(0) = 0$ and $\mathscr{A}(0) = -M(0)\mathscr{B}(0)$. Thus, if $\mathscr{B}(0) = 0$,

$\mathscr{A}(0) = 0$ also. ■

Lemma 4.8 can be used to determine the number of solutions to the problem. We use it to provide a relationship between the output of the FPHPS algorithm and the number of solutions to the power Hermite-Padé approximation problem such that the part of the constant terms corresponding to the vector polynomial reversal are linearly independent over $\mathbb{F}[\lambda]$.

Lemma 4.9. *Let $\mathbb{F}$ be a field, $M \in \mathbb{F}^{\beta_1 \times \beta_2}[\lambda]$, $\nu_2 > 0$, and $\sigma \geq \beta_1$. Let the polynomial tuples $\mathscr{P}_{1,\sigma}, \ldots, \mathscr{P}_{\beta_1+\beta_2,\sigma}$ be the output of the FPHPS algorithm for the power Hermite-Padé approximation problem of type $(\mathscr{N}, \sigma, \beta_1)$ given by the polynomial tuple $\mathscr{F} = (\mathscr{F}^{[1]}, \ldots, \mathscr{F}^{[\beta_1+\beta_2]})$ with*

$$\mathscr{F}^{[i]}(\lambda) = (1, \lambda, \lambda^2, \ldots, \lambda^{\beta_1-1}) \cdot \mathscr{G}^{[i]}(\lambda^{\beta_1})$$

where $\mathscr{G}^{[i]}$ is the i-th column of the matrix polynomial

$$\mathscr{G}(\lambda) = \begin{bmatrix} I & M \end{bmatrix} \in \mathbb{F}^{\beta_1 \times (\beta_1+\beta_2)}[\lambda].$$

For any polynomial tuple $\mathscr{P} = (\mathscr{P}^{[1]}, \ldots, \mathscr{P}^{[\beta_1+\beta_2]})$, let $\mathscr{B}$ be the polynomial sub-tuple $\mathscr{B} = (\mathscr{P}^{[\beta_1+1]}, \ldots, \mathscr{P}^{[\beta_1+\beta_2]})$, and let $L_\sigma = \{1, \ldots, \beta_1 + \beta_2\} \setminus \{\pi_{\sigma-\beta_1}, \ldots, \pi_{\sigma-1}\}$. Then, there exist $k \leq \beta_2$ solutions $\overline{\mathscr{P}}_1, \ldots, \overline{\mathscr{P}}_k$ to the power Hermite-Padé approximation problem such that $\overline{\mathscr{B}}_1(0), \ldots, \overline{\mathscr{B}}_k(0)$ are linearly independent over $\mathbb{F}$ if and only if there are distinct $l_1, \ldots, l_k \in L_\sigma$ with $\operatorname{dct}(\mathscr{P}_{l_j,\sigma}) > 0$. The approximants are given by $\overline{\mathscr{P}}_j = \mathscr{P}_{l_j,\sigma}$

for $1 \leq j \leq k$.

Proof. Clearly, $\overline{\mathscr{P}}_1(0), \ldots, \overline{\mathscr{P}}_k(0)$ are linearly independent over $\mathbb{F}$ if $\overline{\mathscr{B}}_1(0), \ldots, \overline{\mathscr{B}}_k(0)$ are also linearly independent over $\mathbb{F}$. Suppose $\overline{\mathscr{P}}_k(0), \ldots, \overline{\mathscr{P}}_k(0)$ are linearly independent over $\mathbb{F}$. For every $c_1, \ldots, c_k \in \mathbb{F}$, $\sum_{i=1}^{k} c_i \overline{\mathscr{P}}_i(0) \neq 0$. Let $\overline{\mathscr{A}} = (\overline{\mathscr{P}}^{[1]}, \ldots, \overline{\mathscr{P}}^{[\beta_1]})$. If $\overline{\mathscr{B}}_1(0), \ldots, \overline{\mathscr{B}}_k(0)$ are linearly dependent over $\mathbb{F}$, there exist $c_1, \ldots, c_k \in \mathbb{F}$ such that $\sum_{i=1}^{k} c_i \overline{\mathscr{B}}_i(0) = 0$. By Lemma 4.8 on page 108, this means $\sum_{i=1}^{k} c_i \overline{\mathscr{A}}_i(0) = 0$, and, thus, $\sum_{i=1}^{k} c_i \overline{\mathscr{P}}_i(0) = 0$, which contradicts the linear independence of the constant tuples $\overline{\mathscr{P}}_1(0), \ldots, \overline{\mathscr{P}}_k(0)$ over $\mathbb{F}$. Therefore, $\overline{\mathscr{P}}_1(0), \ldots, \overline{\mathscr{P}}_k(0)$ are linearly independent over $\mathbb{F}$ if and only if $\overline{\mathscr{B}}_1(0), \ldots, \overline{\mathscr{B}}_k(0)$ are linearly independent over $\mathbb{F}$.

Using Corollary 4.2 of Beckermann and Labahn (1994), there exist $k \leq \beta_2$ solutions $\overline{\mathscr{P}}_1, \ldots, \overline{\mathscr{P}}_k$ to the power Hermite-Padé approximation problem such that the scalar tuples $\overline{\mathscr{P}}_1(0), \ldots, \overline{\mathscr{P}}_k(0)$ are linearly independent over $\mathbb{F}$ if and only if there exist distinct $l_1, \ldots, l_k \in L_\sigma$ with $\operatorname{dct}(\mathscr{P}_{l_j,\sigma}) > 0$, and the linearly independent approximants are given by $\overline{\mathscr{P}}_j = \mathscr{P}_{l_j,\sigma}$ for $1 \leq j \leq k$. ∎

There can be up to β_2 solutions to the power Hermite-Padé approximation problem such that the constant coefficients of the last β_2 entries of each polynomial tuple are linearly independent over $\mathbb{F}$. Theorem 4.7 on page 102 can then be used to show there are exactly β_2 such solutions. In other words, at least β_2 of the polynomial tuples produced by the FPHPS algorithm must have positive defect by Lemma 4.9. However, we can use the relation between generating vector polynomials and solutions to the power Hermite-Padé approximation problem to show there can be no more the β_2.

Theorem 4.9. *Let $\mathbb{F}$ be a field, and let the matrix sequence $\{B_i\}_{i=0}^{\infty} \in \left(\mathbb{F}^{\beta_1 \times \beta_2}\right)^{\mathbb{Z}_{\geq 0}}$ with $\beta_1, \beta_2 > 0$ be linearly generated. Let $\nu_1 \geq \gamma_1$, $\nu_2 \geq \gamma_2$, and $\sigma = (\nu_1 + \nu_2)\beta_1$. Let the polynomial tuples $\mathscr{P}_{1,\sigma}, \ldots, \mathscr{P}_{\beta_1+\beta_2,\sigma}$ be the output of the FPHPS algorithm for the power Hermite-Padé approximation problem of type $(\mathscr{N}, \sigma, \beta_1)$ given by the multiindex $\mathscr{N} = (\mathscr{N}^{[1]}, \ldots, \mathscr{N}^{[\beta_1+\beta_2]})$ with*

$$\mathscr{N}^{[i]} = \begin{cases} \nu_2 - 1 & \text{if } 1 \leq i \leq \beta_1, \\ \nu_2 & \text{if } \beta_1 + 1 \leq i \leq \beta_1 + \beta_2 \end{cases}$$

and the polynomial tuple $\mathscr{F} = (\mathscr{F}^{[1]}, \ldots, \mathscr{F}^{[\beta_1+\beta_2]})$ with

$$\mathscr{F}^{[i]}(\lambda) = (1, \lambda, \lambda^2, \ldots, \lambda^{\beta_1 - 1}) \cdot \mathscr{G}^{[i]}(\lambda^{\beta_1})$$

where $\mathscr{G}^{[i]}$ is the i-th column of the matrix polynomial

$$\mathscr{G}(\lambda) = \left[I \quad -\left(\sum_{i=0}^{\nu_1+\nu_2-1} B_i \lambda^i\right) \right] \in \mathbb{F}^{\beta_1 \times (\beta_1+\beta_2)}[\lambda].$$

For any polynomial tuple $\mathscr{P} = (\mathscr{P}^{[1]}, \ldots, \mathscr{P}^{[\beta_1+\beta_2]})$, let $\mathscr{B}$ be the polynomial sub-tuple $\mathscr{B} = (\mathscr{P}^{[\beta_1+1]}, \ldots, \mathscr{P}^{[\beta_1+\beta_2]})$. Then, exactly β_2 of the polynomial tuples constructed by the FPHPS algorithm have positive defect. Furthermore, if these approximants with positive defect are denoted $\mathscr{P}_{i_1,\sigma}, \ldots, \mathscr{P}_{i_{\beta_2},\sigma}$, then $\mathscr{B}_{i_1,\sigma}(0), \ldots, \mathscr{B}_{i_{\beta_2},\sigma}(0)$ are linearly independent over $\mathbb{F}$.

Proof. Let $C_1, \ldots, C_{\beta_2}$ be the columns of the minimal generating matrix polynomial F of the matrix sequence $\{B_i\}_{i=0}^{\infty}$. Let $\hat{C}_i$ be the vector polynomial reversal of C_i with respect to degree $d_i = \deg(C_i)$. Then, by Theorem 4.7 on page 102, for each C_i, there exists a vector polynomial $C_i^{(\text{res})}$ of degree at most $d_i - 1$ such that the polynomial tuple

$$\mathscr{P}_i = \begin{bmatrix} C_i^{(\text{res})} \\ \hat{C}_i \end{bmatrix} \in \mathbb{F}^{\beta_1+\beta_2}[\lambda]$$

solves the power Hermite-Padé approximation problem. $\mathscr{P}_i(0) \neq 0$ and $\operatorname{dct}(\mathscr{P}_i) > 0$. Because F is in Popov form, the leading coefficients of $C_1, \ldots, C_{\beta_2}$ are linearly independent over $\mathbb{F}$. Thus, $\hat{C}_1(0), \ldots, \hat{C}_{\beta_2}(0)$, and $\mathscr{P}_1(0), \ldots, \mathscr{P}_{\beta_2}(0)$, are also linearly independent over $\mathbb{F}$ and $\mathscr{P}_1, \ldots, \mathscr{P}_{\beta_2}$ are linearly independent over $\mathbb{F}[\lambda]$. By Lemma 4.9 on page 110, β_2 of the polynomial tuples $\mathscr{P}_{i_1,\sigma}, \ldots, \mathscr{P}_{i_{\beta_2},\sigma}$ found by the FPHPS algorithm have positive defect and such that $\mathscr{B}_{i_1,\sigma}(0), \ldots, \mathscr{B}_{i_{\beta_2},\sigma}(0)$ are linearly independent over $\mathbb{F}$.

Suppose there exists another approximant $\mathscr{P} = (\mathscr{P}^{[1]}, \ldots, \mathscr{P}^{[\beta_1+\beta_2]})$ that is linearly independent from $\mathscr{P}_1, \ldots, \mathscr{P}_{\beta_2}$ over $\mathbb{F}[\lambda]$. Let $\mathscr{B} = (\mathscr{P}^{[\beta_1+1]}, \ldots, \mathscr{P}^{[\beta_1+\beta_2]})$, and let $\hat{\mathscr{B}}$ be the reversal of $\mathscr{B}$ with respect to degree $d = \nu_2 + 1 - \operatorname{dct}(\mathscr{P})$. By Theorem 4.8 on page 105, $\hat{\mathscr{B}}$ generates the matrix sequence $\{B_i\}_{i=0}^{\infty}$, which means $\hat{\mathscr{B}}$ must be a linear combination of $C_1, \ldots, C_{\beta_2}$ over $\mathbb{F}[\lambda]$,

$$\hat{\mathscr{B}} = \sum_{i=1}^{\beta_2} g_i \, C_i$$

where $g_i \in \mathbb{F}[\lambda]$ for $1 \leq i \leq \beta_2$. Thus,

$$\mathscr{B} = \mathrm{rev}_d(\hat{\mathscr{B}}) = \lambda^d \sum_{i=1}^{\beta_2} g_i\left(\frac{1}{\lambda}\right)\ C_i\left(\frac{1}{\lambda}\right) = \sum_{i=1}^{\beta_2} \lambda^{d-d_i} g_i\left(\frac{1}{\lambda}\right)\ \hat{C}_i(\lambda).$$

The leading coefficients of $C_1, \ldots, C_{\beta_2}$ are linearly independent over $\mathbb{F}$, which means

$$\deg(\hat{\mathscr{B}}) = \max_{1\leq i\leq \beta_2} \{\deg(g_i) + \deg(C_i)\} = \max_{1\leq i\leq \beta_2} \{\deg(g_i) + d_i\}$$

by Lemma 4.3 on page 79 and

$$\deg(g_i) + d_i \leq \deg(\hat{\mathscr{B}}) \leq d, \qquad 1 \leq i \leq \beta_2.$$

Thus,

$$\deg(g_i) \leq d - d_i, \qquad 1 \leq i \leq \beta_2,$$

and

$$\lambda^{d-d_i} g_i\left(\frac{1}{\lambda}\right) \in \mathbb{F}[\lambda], \qquad 1 \leq i \leq \beta_2.$$

Therefore, $\mathscr{B}$ is a linear combination of $\hat{C}_1, \ldots, \hat{C}_{\beta_2}$ over $\mathbb{F}[\lambda]$.

Let $\mathscr{P}_0$ be the polynomial tuple

$$\mathscr{P}_0 = \mathscr{P} - \sum_{i=1}^{\beta_2} \lambda^{d-d_i} g_i\left(\frac{1}{\lambda}\right)\ \mathscr{P}_i(\lambda).$$

$\mathscr{P}_0$ solves the power Hermite-Padé approximation problem, and

$$\mathscr{B}_0 = \mathscr{B} - \sum_{i=1}^{\beta_2} \lambda^{d-d_i} g_i \left(\frac{1}{\lambda}\right) \hat{C}_i(\lambda) = 0^{\beta_2}.$$

Then, by Lemma 4.8 on page 108, $\mathscr{P}_0 = 0^{\beta_1+\beta_2}$, and $\mathscr{P}$ is not linearly independent over $\mathbb{F}[\lambda]$ from $\mathscr{P}_1, \ldots, \mathscr{P}_{\beta_2}$, which is a contradiction. Therefore, there cannot be more than β_2 solutions to the power Hermite-Padé approximation problem that are linearly independent over $\mathbb{F}[\lambda]$, and, thus, only β_2 of the approximants $\mathscr{P}_{1,\sigma}, \ldots, \mathscr{P}_{\beta_1+\beta_2,\sigma}$ have positive defect (Beckermann and Labahn, 1994, Corollary 4.2). ■

Let $\mathscr{P}_{i_1,\sigma}, \ldots, \mathscr{P}_{i_{\beta_2},\sigma}$ be the polynomial tuples found by the FPHPS algorithm that have positive defect. The reversals of $\mathscr{B}_{i_1,\sigma}, \ldots, \mathscr{B}_{i_{\beta_2},\sigma}$ are linearly independent over $\mathbb{F}[\lambda]$, and, by Theorem 4.8 on page 105, they generate the matrix sequence. We can use a proof similar to that of Theorem 4.9 to show that they span the generating vector polynomials over $\mathbb{F}[\lambda]$. However, we first need to show that the leading coefficients of the output polynomial tuples are linearly independent over $\mathbb{F}$. The following lemma shows this linear independence is maintained throughout the FPHPS algorithm.

Lemma 4.10. *Let $\mathbb{F}$ be a field, $\beta_1, \beta_2 > 0$, $\nu_2 > 0$, $\mathscr{F} = (\mathscr{F}^{[1]}, \ldots, \mathscr{F}^{[\beta_1+\beta_2]}) \in \mathbb{F}^m[\lambda]$ be a polynomial tuple, and $\mathscr{N} = (\mathscr{N}^{[1]}, \ldots, \mathscr{N}^{[\beta_1+\beta_2]})$ be the multiindex given by*

$$\mathscr{N}^{[i]} = \begin{cases} \nu_2 - 1 & \text{if } 1 \le i \le \beta_1, \\ \nu_2 & \text{if } \beta_1 + 1 \le i \le \beta_1 + \beta_2, \end{cases}.$$

Then, the leading coefficients of $\mathscr{P}_{1,\sigma}, \ldots, \mathscr{P}_{\beta_1+\beta_2,\sigma}$ *are linearly independent over* $\mathbb{F}$ *for all* $\sigma \geq 0$ *in the FPHPS algorithm.*

Proof. The proof is by induction. The leading coefficients of $\mathscr{P}_{1,0}, \ldots, \mathscr{P}_{\beta_1+\beta_2,0}$ are linearly independent over $\mathbb{F}$ by construction.

Suppose the leading coefficients $\mathrm{lc}(\mathscr{P}_{1,\sigma}), \ldots, \mathrm{lc}(\mathscr{P}_{\beta_1+\beta_2,\sigma})$ are linearly independent over $\mathbb{F}$. If $\Lambda_\sigma = \{\}$, the leading coefficients $\mathrm{lc}(\mathscr{P}_{1,\sigma+1}), \ldots, \mathrm{lc}(\mathscr{P}_{\beta_1+\beta_2,\sigma+1})$ are also linearly independent over $\mathbb{F}$ since $\mathrm{lc}(\mathscr{P}_{i,\sigma+1}) = \mathrm{lc}(\mathscr{P}_{i,\sigma})$ for $1 \leq i \leq \beta_1 + \beta_2$.

If $\Lambda_\sigma \neq \{\}$ and $l \in \Lambda_\sigma$, $l \neq \pi = \pi_\sigma$, then $\mathrm{dct}(\mathscr{P}_{\pi,\sigma}) \geq \mathrm{dct}(\mathscr{P}_{l,\sigma})$, $|\mathscr{N}^{[\pi]} - \mathscr{N}^{[l]}| \leq 1$, and

$$\begin{aligned}
\deg(\mathscr{P}_{l,\sigma}) &= \mathscr{N}^{[l]} + 1 - \mathrm{dct}(\mathscr{P}_{l,\sigma}) \\
&= \mathscr{N}^{[\pi]} + 1 - \mathrm{dct}(\mathscr{P}_{\pi,\sigma}) + ((\mathrm{dct}(\mathscr{P}_{\pi,\sigma}) - \mathrm{dct}(\mathscr{P}_{l,\sigma})) - (\mathscr{N}^{[\pi]} - \mathscr{N}^{[l]})) \\
&= \deg(\mathscr{P}_{\pi,\sigma}) + ((\mathrm{dct}(\mathscr{P}_{\pi,\sigma}) - \mathrm{dct}(\mathscr{P}_{l,\sigma})) - (\mathscr{N}^{[\pi]} - \mathscr{N}^{[l]})).
\end{aligned}$$

If $\mathrm{dct}(\mathscr{P}_{\pi,\sigma}) = \mathrm{dct}(\mathscr{P}_{l,\sigma})$, then $l > \pi$ and $\mathscr{N}^{[l]} \leq \mathscr{N}^{[\pi]}$, which means $\deg(\mathscr{P}_{l,\sigma}) \geq \deg(\mathscr{P}_{\pi,\sigma})$. If $\mathrm{dct}(\mathscr{P}_{\pi,\sigma}) > \mathrm{dct}(\mathscr{P}_{l,\sigma})$, then $(\mathrm{dct}(\mathscr{P}_{\pi,\sigma}) - \mathrm{dct}(\mathscr{P}_{l,\sigma})) - (\mathscr{N}^{[\pi]} - \mathscr{N}^{[l]}) \geq 0$, and $\deg(\mathscr{P}_{l,\sigma}) \geq \deg(\mathscr{P}_{\pi,\sigma})$.

Thus, for every $1 \leq i \leq \beta_1 + \beta_2$,

$$\mathrm{lc}(\mathscr{P}_{i,\sigma+1}) = \begin{cases} \mathrm{lc}(\mathscr{P}_{i,\sigma}) - \frac{c_{i,\sigma}}{c_{\pi,\sigma}} \mathrm{lc}(\mathscr{P}_{\pi,\sigma}) & \text{if } i \in \Lambda_\sigma,\ \deg(\mathscr{P}_{l,\sigma}) = \deg(\mathscr{P}_{\pi,\sigma}), \\ \mathrm{lc}(\mathscr{P}_{i,\sigma}) & \text{otherwise,} \end{cases}$$

and linear independence over $\mathbb{F}$ is preserved. ■

The linear independence of the leading coefficients over $\mathbb{F}$ leads to a basis for the generating vector polynomials over $\mathbb{F}[\lambda]$. The proof is similar to that of Theorem 4.9 on page 112.

Theorem 4.10. *Let $\mathbb{F}$ be a field, and let the matrix sequence $\{B_i\}_{i=0}^{\infty} \in \left(\mathbb{F}^{\beta_1 \times \beta_2}\right)^{\mathbb{Z}_{\geq 0}}$ with $\beta_1, \beta_2 > 0$ be linearly generated. Let $\nu_1 \geq \gamma_1$, $\nu_2 \geq \gamma_2$, and $\sigma = (\nu_1 + \nu_2)\beta_1$. Let the polynomial tuples $\mathscr{P}_{1,\sigma}, \ldots, \mathscr{P}_{\beta_1+\beta_2,\sigma}$ be the output of the FPHPS algorithm for the power Hermite-Padé approximation problem of type $(\mathscr{N}, \sigma, \beta_1)$ given by the multiindex $\mathscr{N} = (\mathscr{N}^{[1]}, \ldots, \mathscr{N}^{[\beta_1+\beta_2]})$ with*

$$\mathscr{N}^{[i]} = \begin{cases} \nu_2 - 1 & \text{if } 1 \leq i \leq \beta_1, \\ \nu_2 & \text{if } \beta_1 + 1 \leq i \leq \beta_1 + \beta_2 \end{cases}$$

and the polynomial tuple $\mathscr{F} = (\mathscr{F}^{[1]}, \ldots, \mathscr{F}^{[\beta_1+\beta_2]})$ with

$$\mathscr{F}^{[i]}(\lambda) = (1, \lambda, \lambda^2, \ldots, \lambda^{\beta_1 - 1}) \cdot \mathscr{G}^{[i]}(\lambda^{\beta_1})$$

where $\mathscr{G}^{[i]}$ is the i-th column of the matrix polynomial

$$\mathscr{G}(\lambda) = \left[I \quad -\left(\sum_{i=0}^{\nu_1+\nu_2-1} B_i \lambda^i\right) \right] \in \mathbb{F}^{\beta_1 \times (\beta_1+\beta_2)}[\lambda].$$

Then, exactly β_2 of the polynomial tuples $\mathscr{P}_{1,\sigma}, \ldots, \mathscr{P}_{\beta_1+\beta_2,\sigma}$ have positive defect. Let

these be denoted $\mathscr{P}_{i_1,\sigma}, \ldots, \mathscr{P}_{i_{\beta_2},\sigma}$ *and let* $d_j = \nu_2 + 1 - \operatorname{dct}(\mathscr{P}_{i_j,\sigma})$. *Then, the vector polynomial reversals*

$$b_j = \operatorname{rev}_{d_j}\left(\begin{bmatrix} \mathscr{P}_{i_j,\sigma}{}^{[\beta_1+1]} & \ldots & \mathscr{P}_{i_j,\sigma}{}^{[\beta_1+\beta_2]} \end{bmatrix}^T\right) \in \mathbb{F}^{\beta_2}[\lambda], \qquad 1 \le j \le \beta_2,$$

form a basis over $\mathbb{F}[\lambda]$ *for the module of right generating vector polynomials of the matrix sequence* $\{B_i\}_{i=0}^{\infty}$, *their leading coefficient vectors are linearly independent over* $\mathbb{F}$, $\deg(b_j) = d_j$ *for* $1 \le j \le \beta_2$, *and*

$$\deg\left(\det\left(\begin{bmatrix} b_1 & \ldots & b_{\beta_2} \end{bmatrix}\right)\right) = \sum_{j=1}^{\beta_2} d_j = \beta_2(\nu_2+1) - \sum_{j=1}^{\beta_2} \operatorname{dct}(\mathscr{P}_{i_j,\sigma}).$$

Proof. Let $\mathscr{B}_{i,\sigma}$ be the polynomial sub-tuple $\mathscr{B}_{i,\sigma} = (\mathscr{P}_{i,\sigma}{}^{[\beta_1+1]}, \ldots, \mathscr{P}_{i,\sigma}{}^{[\beta_1+\beta_2]})$. The polynomial tuples $\mathscr{P}_{i_1,\sigma}, \ldots, \mathscr{P}_{i_{\beta_2},\sigma}$ exist and $\mathscr{B}_{i_1,\sigma}(0), \ldots, \mathscr{B}_{i_{\beta_2},\sigma}(0)$ are linearly independent over $\mathbb{F}$ by Theorem 4.9 on page 112. This means

$$\deg(b_j) = d_j = \nu_2 + 1 - \operatorname{dct}(\mathscr{P}_{i_j,\sigma}) \le \deg(\mathscr{P}_{i_j,\sigma}), \qquad 1 \le j \le \beta_2,$$

and the leading coefficients of $b_1, \ldots, b_{\beta_2}$ are linearly independent over $\mathbb{F}$. Thus,

$$\deg\left(\det\left(\begin{bmatrix} b_1 & \ldots & b_{\beta_2} \end{bmatrix}\right)\right) = \sum_{j=1}^{\beta_2} \deg(b_j) = \sum_{j=1}^{\beta_2} d_j = \beta_2(\nu_2+1) - \sum_{j=1}^{\beta_2} \operatorname{dct}(\mathscr{P}_{i_j,\sigma}).$$

By Theorem 4.8 on page 105, $b_1, \ldots, b_{\beta_2}$ generate the matrix sequence.

Let $C = \sum_{i=0}^{d} C[i]\lambda^i$ be a right generating vector polynomial for the matrix sequence

with degree d. By Theorem 4.7 on page 102, the vector polynomial reversal $\hat{C}$ of C with respect to degree d and a vector polynomial $C^{(\mathrm{res})}$ of degree at most $d-1$ form a solution

$$\mathscr{P} = \begin{bmatrix} C^{(\mathrm{res})} \\ \hat{C} \end{bmatrix} \in \mathbb{F}^{\beta_1+\beta_2}[\lambda]$$

to the power Hermite-Padé approximation problem with degree $\deg(\mathscr{P}) \leq d$. There exist polynomials $g_1, \ldots, g_{\beta_2} \in \mathbb{F}[\lambda]$ with $\deg(g_j) \leq \mathrm{dct}(\mathscr{P}_{i_j,\sigma})$ for $1 \leq i \leq \beta_2$ such that $\mathscr{P} = \sum_{j=1}^{\beta_2} g_j \mathscr{P}_{i_j\sigma}$. If $\hat{b}_j$ is the vector polynomial reversal of b_j with respect to degree d_j, then $\hat{C} = \sum_{j=1}^{\beta_2} g_j \hat{b}_j$ and

$$C(\lambda) = \lambda^d \hat{C}\left(\frac{1}{\lambda}\right) = \lambda^d \sum_{i=j}^{\beta_2} g_j\left(\frac{1}{\lambda}\right) \hat{b}_j\left(\frac{1}{\lambda}\right) = \sum_{j=1}^{\beta_2} \lambda^{d-d_i} g_j\left(\frac{1}{\lambda}\right) b_j(\lambda).$$

By Lemma 4.10 on page 115, the leading coefficients of $\mathscr{P}_{i_1,\sigma}, \ldots, \mathscr{P}_{i_{\beta_2},\sigma}$ are linearly independent over $\mathbb{F}$, and

$$\deg(\mathscr{P}) = \max_{1\leq j\leq \beta_2} \{\deg(g_j) + \deg(\mathscr{P}_{i_j,\sigma})\}$$

by Lemma 4.3 on page 79. Thus,

$$\deg(\mathscr{P}) \geq \deg(g_j) + \deg(\mathscr{P}_{i_j,\sigma}), \qquad 1 \leq j \leq \beta_2,$$

and

$$\deg(g_j) \leq \deg(\mathscr{P}) - \deg(\mathscr{P}_{i_j,\sigma}) \leq \deg(\mathscr{P}) - d_j \leq d - d_j, \qquad 1 \leq j \leq \beta_2.$$

Therefore,

$$\lambda^{d-d_j} g_j \in \mathbb{F}[\lambda], \qquad 1 \leq j \leq \beta_2,$$

and C is a linear combination of $b_1, \ldots, b_{\beta_2}$ over $\mathbb{F}[\lambda]$. ■

The update mechanism in the FPHPS algorithm ensures that, if $\nu_1 \geq \gamma_1$ and $\nu_2 \geq \gamma_2$, stopping the algorithm at $\sigma < (\nu_1+\nu_2)\beta_1$ will result in either too many polynomial tuples with positive defect or the determinantal degree of the potential basis elements described in Theorem 4.10 will be no larger than prescribed at $\sigma = (\nu_1+\nu_2)\beta_1$. This is because the algorithm iterates over the order σ of the power Hermite-Padé approximation problem, and at every σ it chooses π to be the index of the polynomial tuple with largest defect that does not have order σ. The algorithm uses this polynomial tuple to update the others. This polynomial tuple is the only one to have its defect changed, and it will always decrease. Thus, once the algorithm has only β_2 polynomial tuples with positive defect remaining, either these polynomial tuples must designate a basis for $\mathbb{G}$ as described in Theorem 4.10 or the algorithm must eventually decrease the defect of at least one of them. As shown in Theorem 4.10, decreasing the defect of one of the polynomial tuples will increase the determinantal degree of the matrix whose columns are the basis elements.

In fact, suppose the algorithm produces only β_2 polynomial tuples, $\mathscr{P}_{i_1,\sigma}, \ldots, \mathscr{P}_{i_{\beta_2},\sigma}$,

with positive defect and order $\sigma < (\nu_1 + \nu_2)\beta_1$. If all of these polynomial tuples have order $\sigma + 1$, then

$$\mathscr{P}_{i_j,\sigma+1} = \mathscr{P}_{i_j,\sigma}, \qquad 1 \leq j \leq \beta_2.$$

If one of these polynomial tuples does not have order $\sigma+1$, the update mechanism requires that there exists a j with $1 \leq j \leq \beta_2$ such that $\pi = i_j$, and $\mathscr{P}_{i_j,\sigma+1} = \lambda\mathscr{P}_{i_j,\sigma}$. This means $\mathscr{B}_{i_j,\sigma+1}(0) = 0^{\beta_2}$, and thus $\mathscr{B}_{i_1,\sigma+1}(0), \ldots, \mathscr{B}_{i_{\beta_2},\sigma+1}(0)$ are not linearly independent over $\mathbb{F}$. By induction, $\mathscr{B}_{i_1,(\nu_1+\nu_2)\beta_1}(0), \ldots, \mathscr{B}_{i_{\beta_2},(\nu_1+\nu_2)\beta_1}(0)$ cannot be linearly independent over $\mathbb{F}$, which contradicts Theorem 4.9 on page 112. Therefore, if $\nu_1 \geq \gamma_1$ and $\nu_2 \geq \gamma_2$, once the FPHPS algorithm has produced only β_2 polynomial tuples, $\mathscr{P}_{i_1,\sigma}, \ldots, \mathscr{P}_{i_{\beta_2},\sigma}$, with positive defect and order $\sigma < (\nu_1 + \nu_2)\beta_1$, these β_2 polynomial tuples have order $(\nu_1 + \nu_2)\beta_1$ and are thus the polynomial tuples computed by the FPHPS algorithm for $\sigma = (\nu_1 + \nu_2)\beta_1$,

$$\mathscr{P}_{i_j,(\nu_1+\nu_2)\beta_1} = \mathscr{P}_{i_j,\sigma}, \qquad 1 \leq j \leq \beta_2.$$

On the other hand, if $\nu_2 < \gamma_2$, the FPHPS algorithm will return fewer than β_2 polynomial tuples with positive defect since any basis over $\mathbb{F}[\lambda]$ for the module of right generating vector polynomials must contain at least element of degree no less than γ_2. Unfortunately, stopping the algorithm at $\sigma < (\nu_1 + \nu_2)\beta_1$ may result in β_2 polynomial tuples with positive defect.

Theorem 4.10 on page 117 gives method for computing the minimal generating matrix polynomial for any linearly generated matrix sequence $\{B_i\}_{i=0}^{\infty}$. First, compute the polynomial tuple $\mathscr{F} = (\mathscr{F}^{[1]}, \ldots, \mathscr{F}^{[\beta_1+\beta_2]})$ with $\mathscr{F}^{[i]}(\lambda) = (1, \lambda, \lambda^2, \ldots, \lambda^{\beta_1-1}) \cdot \mathscr{G}^{[i]}(\lambda^{\beta_1})$

where $\mathscr{G}^{[i]}$ is the i-th column of the matrix polynomial

$$\mathscr{G}(\lambda) = \left[I \quad -\left(\sum_{i=0}^{\nu_1+\nu_2-1} B_i \lambda^i\right) \right] \in \mathbb{F}^{\beta_1 \times (\beta_1+\beta_2)}[\lambda].$$

Then, call the Beckermann-Labahn FPHPS algorithm with $\mathscr{F}$, $s = \beta_1$, $\sigma = (\gamma_1 + \gamma_2)\beta_1$, and multiindex $\mathscr{N} = (\mathscr{N}^{[1]}, \ldots, \mathscr{N}^{[\beta_1+\beta_2]})$ with

$$\mathscr{N}^{[i]} = \begin{cases} \nu_2 - 1 & \text{if } 1 \le i \le \beta_1, \\ \nu_2 & \text{if } \beta_1 + 1 \le i \le \beta_1 + \beta_2. \end{cases}$$

The last β_2 entries of the β_2 vector polynomial reversals with positive defects form a basis for the module of right generating vector polynomials of the matrix sequence. The minimal generating matrix polynomial is the Popov form of the matrix whose columns are these basis vectors.

For example, consider Example 4.1 on page 83 with the input to the FPHPS algorithm $\sigma = (2+2)2 = 8$,

$$\mathscr{F} = \begin{bmatrix} 1 \\ \lambda \\ -\lambda - \lambda^2 - 3\lambda^3 - \lambda^4 - 4\lambda^5 - 2\lambda^6 - 7\lambda^7 \\ -1 - 2\lambda - \lambda^3 - \lambda^4 - 4\lambda^5 - \lambda^6 - 4\lambda^7 \end{bmatrix}^{\mathrm{T}},$$

and $\mathscr{N} = (1, 1, 2, 2)$. The algorithm produces the polynomial tuples

$$\mathscr{P}_{1,8} = \begin{bmatrix} \lambda^3 \\ 0 \\ -2\lambda^3 \\ \lambda^3 \end{bmatrix}, \ \mathscr{P}_{2,8} = \begin{bmatrix} \frac{1}{2}\lambda^3 \\ \lambda^3 \\ 0 \\ \frac{1}{2}\lambda^3 \end{bmatrix}, \ \mathscr{P}_{3,8} = \begin{bmatrix} -1 \\ -2 \\ \lambda \\ -1 \end{bmatrix}, \ \mathscr{P}_{4,8} = \begin{bmatrix} 1 \\ 1 \\ -1 \\ \lambda + 1 \end{bmatrix},$$

and their defects

$$\mathrm{dct}(\mathscr{P}_{1,8}) = -1, \ \mathrm{dct}(\mathscr{P}_{2,8}) = -1, \ \mathrm{dct}(\mathscr{P}_{3,8}) = 2, \ \text{and} \ \mathrm{dct}(\mathscr{P}_{4,8}) = 2.$$

Thus, the last two entries of the vector polynomial reversals of $\mathscr{P}_{1,8}$ and $\mathscr{P}_{2,8}$ form the columns of the matrix

$$G = \begin{bmatrix} 1 & -\lambda \\ -\lambda & \lambda + 1 \end{bmatrix}$$

whose Popov form is the minimal generating matrix polynomial

$$F = \begin{bmatrix} \lambda - 1 & -1 \\ -1 & \lambda \end{bmatrix} = G \begin{bmatrix} -1 & -1 \\ -1 & 0 \end{bmatrix}.$$

4.3 Block Wiedemann and Krylov Sequences

In Sections 4.1 and 4.2, we considered any linearly generated matrix sequences $\{B_i\}_{i=0}^{\infty}$. In general, we do not know any bounds on the γ_1 and γ_2 or the determinantal degree

of F. However, the block Wiedemann and block Krylov sequences, $\{X^{\mathrm{T}}A^iY\}_{i=0}^{\infty}$ and $\{A^iY\}_{i=0}^{\infty}$, respectively, arising from the block Wiedemann method are special, and we can find upper bounds for general block projections X and Y. These bounds follow from relations on the minimal generating matrix polynomials for the two sequences, $F^{A,Y}$ and $F_X^{A,Y}$, and the minimal polynomial f^A of the matrix A.

We first show the block Wiedemann and block Krylov sequences are linearly generated by the minimal polynomial f^A of the matrix A.

Lemma 4.11. *Let $\mathbb{F}$ be a field, $A \in \mathbb{F}^{n\times n}$, $X \in \mathbb{F}^{n\times\beta_1}$, and $Y \in \mathbb{F}^{n\times\beta_2}$ with $1 \le \beta_1, \beta_2 \le n$. Then, the block Wiedemann and block Krylov sequences, $\{X^TA^iY\}_{i=0}^{\infty}$ and $\{A^iY\}_{i=0}^{\infty}$, respectively, are linearly generated by the minimal polynomial f^A of the matrix A.*

Proof. The minimal polynomial $f^A = \sum_{i=0}^{m} f^A[i]\lambda^i$ generates the matrix power sequence $\{A^i\}_{i=0}^{\infty}$,

$$\sum_{i=0}^{m} f^A[i]A^{i+j} = 0^{n\times n}, \qquad \forall j \ge 0.$$

Multiplying this equation on the right by the block projection Y gives the equation

$$\sum_{i=0}^{m} f^A[i]A^{i+j}Y = 0^{n\times\beta_2}, \qquad \forall j \ge 0,$$

and thus f^A generates the block Krylov sequence. Similarly,

$$\sum_{i=0}^{m} f^A[i]X^{\mathrm{T}}A^{i+j}Y = 0^{\beta_1\times\beta_2}, \qquad \forall j \ge 0,$$

and f^A generates the block Wiedemann sequence. ■

Because the block Wiedemann and block Krylov sequences are linearly generated, all of the results of Sections 4.1 and 4.2 hold for these matrix sequences. In particular, the existence of the modules of right generating vector polynomials for the sequences over $\mathbb{F}[\lambda]$ and the minimal generating matrix polynomials $F_X^{A,Y}$ and $F^{A,Y}$. We now show that the module over $\mathbb{F}[\lambda]$ of the right generating vector polynomials of the block Krylov sequence $\{A^iY\}_{i=0}^{\infty}$ is a sub-module of the module of right generating vector polynomials of the block Wiedemann sequence $\{X^{\mathrm{T}}A^iY\}_{i=0}^{\infty}$. This result will be used to relate $F_X^{A,Y}$ and $F^{A,Y}$.

Lemma 4.12. *Let $\mathbb{F}$ be a field, $A \in \mathbb{F}^{n\times n}$, $X \in \mathbb{F}^{n\times\beta_1}$, and $Y \in \mathbb{F}^{n\times\beta_2}$ with $1 \leq \beta_1, \beta_2 \leq n$. Then, the module $\mathbb{G}^{A,Y}$ of right generating vector polynomials of the block Krylov sequence $\{A^iY\}_{i=0}^{\infty}$ is a sub-module of the module $\mathbb{G}_X^{A,Y}$ of right generating vector polynomials of the block Wiedemann sequence $\{X^TA^iY\}_{i=0}^{\infty}$.*

Proof. By Lemma 4.1 on page 69, $\mathbb{G}^{A,Y}$ and $\mathbb{G}_X^{A,Y}$ are both sub-modules of the $\mathbb{F}[\lambda]$-module $\mathbb{F}^{\beta_2}[\lambda]$, so we only need to show that $\mathbb{G}^{A,Y}$ is contained within $\mathbb{G}_X^{A,Y}$.

Let $C = \sum_{i=0}^{d} C[i]\lambda^i$ be a right generating vector polynomial for the block Krylov sequence. Then,

$$\sum_{i=0}^{d} A^{i+j}YC[i] = 0^n, \qquad \forall j \geq 0,$$

and

$$\sum_{i=0}^{d} X^{\mathrm{T}}A^{i+j}YC[i] = 0^{\beta_1}, \qquad \forall j \geq 0.$$

Thus, C is a right generating vector polynomial for the block Wiedemann sequence, and $\mathbb{G}^{A,Y} \subset \mathbb{G}_X^{A,Y}$. ∎

Any vector polynomial that generates the block Krylov sequence $\{A^iY\}_{i=0}^{\infty}$ must also generate the block Wiedemann sequence $\{X^{\mathrm{T}}A^iY\}_{i=0}^{\infty}$. Thus, $F^{A,Y}$ generates the block Wiedemann sequence, and we have the following relations between $F_X^{A,Y}$ and $F^{A,Y}$. (Recall the definition of the invariant factors of a polynomial matrix from Definition 3.1 on page 45.)

Theorem 4.11. *Let $\mathbb{F}$ be a field, $A \in \mathbb{F}^{n\times n}$, $X \in \mathbb{F}^{n\times\beta_1}$, and $Y \in \mathbb{F}^{n\times\beta_2}$ with $1 \le \beta_1, \beta_2 \le n$. Let N be the sum of the degrees of the β_2 largest invariant factors of the characteristic matrix $\lambda I - A$,*

$$N = \sum_{i=0}^{\beta_2-1} \deg(s_{n-i}\,(\lambda I - A)),$$

Then, $F^{A,Y}$ generates the block Wiedemann sequence $\{X^TA^iY\}_{i=0}^{\infty}$ from the right,

$$\deg(\det(F_X^{A,Y})) \le \deg(\det(F^{A,Y})) \le N \le n,$$

and

$$\deg(F_X^{A,Y}) \le \deg(F^{A,Y}) \le \deg(f^A) \le n,$$

where f^A is the minimal polynomial of the matrix A.

Proof. By Lemma 4.12 on the last page, the columns of $F^{A,Y}$ must generate the block

Wiedemann sequence $\{X^{\mathrm{T}}A^iY\}_{i=0}^{\infty}$, so $F^{A,Y}$ must also generate $\{X^{\mathrm{T}}A^iY\}_{i=0}^{\infty}$. By Theorem 4.3 on page 81, $\deg(\det(F_X^{A,Y})) \leq \deg(\det(F^{A,Y}))$ and $\deg(F_X^{A,Y}) \leq \deg(F^{A,Y})$. Because the block Krylov sequence is linearly generated by the minimal polynomial f^A of the matrix A,

$$\deg(F^{A,Y}) \leq \deg(f^A) \leq n$$

by Corollary 4.4 on page 83.

For the rest of the proof, consider first the invariant factors of the minimal generating matrix polynomial $F^{A,Y}$ and the characteristic matrix $\lambda I - A$ of the matrix A. The i-th largest invariant factor of $F^{A,Y}$ divides the i-th largest invariant factor of $\lambda I - A$ (Kaltofen and Villard, 2001, Theorem 1; 2002), so

$$\deg(s_{\beta_2-i}\left(F^{A,Y}\right)) \leq \deg(s_{n-i}\left(\lambda I - A\right)), \qquad 0 \leq i \leq \beta_2 - 1,$$

and, thus,

$$\deg(\det(F^{A,Y})) = \sum_{i=0}^{\beta_2-1} \deg(s_{\beta_2-i}\left(F^{A,Y}\right)) \leq \sum_{i=0}^{\beta_2-1} \deg(s_{n-i}\left(\lambda I - A\right)) = N \leq n. \qquad \blacksquare$$

Theorem 4.11 gives an upper bound for the degree of the minimal generating matrix polynomials and, thus, for γ_2. Theorem 4.4 on page 87 says $\gamma_1 \leq \deg(f^A) \leq n$ for both sequences. This is the best bound we can find for the block Wiedemann sequence given a general block left projection X. However, we can find a much better bound for the block Krylov sequence. To do so, we again turn to the block Hankel matrices.

Just as we denoted the minimal generating matrix polynomials of the block Wiedemann and Krylov sequences as $F_X^{A,Y}$ and $F^{A,Y}$, respectively, we will denote the block Hankel matrices constructed from the block Wiedemann and block Krylov sequences as $H_X^{A,Y}(\nu_1, \nu_2)$ and $H^{A,Y}(\nu_1, \nu_2)$. In other words,

$$H_X^{A,Y}(\nu_1, \nu_2) = \begin{bmatrix} X^{\mathrm{T}}Y & X^{\mathrm{T}}AY & \cdots & X^{\mathrm{T}}A^{\nu_2-1}Y \\ X^{\mathrm{T}}AY & X^{\mathrm{T}}A^2Y & \cdots & X^{\mathrm{T}}A^{\nu_2}Y \\ \vdots & \vdots & \ddots & \vdots \\ X^{\mathrm{T}}A^{\nu_1-1}Y & X^{\mathrm{T}}A^{\nu_1}Y & \cdots & X^{\mathrm{T}}A^{\nu_1+\nu_2-2}Y \end{bmatrix} \in \mathbb{F}^{\beta_1\nu_1 \times \beta_2\nu_2}$$

and

$$H^{A,Y}(\nu_1, \nu_2) = \begin{bmatrix} Y & AY & \cdots & A^{\nu_2-1}Y \\ AY & A^2Y & \cdots & A^{\nu_2}Y \\ \vdots & \vdots & \ddots & \vdots \\ A^{\nu_1-1}Y & A^{\nu_1}Y & \cdots & A^{\nu_1+\nu_2-2}Y \end{bmatrix} \in \mathbb{F}^{n\nu_1 \times \beta_2\nu_2}$$

for all $\nu_1, \nu_2 > 0$. We will use these block Hankel matrices to determine a bound for $\overline{\gamma_1}$ for the block Krylov sequence.

Theorem 4.12. *Let $\mathbb{F}$ be a field, $A \in \mathbb{F}^{n\times n}$, and $Y \in \mathbb{F}^{n\times\beta_2}$ with $1 \leq \beta_2 \leq n$. Then,* $\operatorname{rank}(H^{A,Y}(\nu_1, \gamma_2 + 1)) = \operatorname{rank}(H^{A,Y}(1, \gamma_2 + 1))$ *for all $\nu_1 \geq 1$. In other words, $\overline{\gamma_1} = 1$.*

Proof. Consider the block Hankel matrix $H^{A,Y}(\nu_1, \gamma_2 + 1)$ for $\nu_1 \geq 1$,

$$H^{A,Y}(\nu_1, \gamma_2 + 1) = \begin{bmatrix} I \\ A \\ \vdots \\ A^{\nu_1 - 1} \end{bmatrix} \begin{bmatrix} Y & AY & \cdots & A^{\gamma_2} Y \end{bmatrix}.$$

The left of these two matrices has full rank,

$$\operatorname{rank} \left(\begin{bmatrix} I \\ A \\ \vdots \\ A^{\nu_1 - 1} \end{bmatrix} \right) = n, \qquad \nu_1 \geq 1$$

so, by Sylvester's inequality (Gantmacher, 1977, page 66),

$$\begin{aligned} n + \operatorname{rank} \left(\begin{bmatrix} Y & AY & \cdots & A^{\gamma_2} Y \end{bmatrix} \right) - n &\leq \operatorname{rank}(H^{A,Y}(\nu_1, \gamma_2 + 1)) \\ &\leq \operatorname{rank} \left(\begin{bmatrix} Y & AY & \cdots & A^{\gamma_2} Y \end{bmatrix} \right) \end{aligned}$$

for all $\nu_1 \geq 1$. Thus, $\operatorname{rank}(H^{A,Y}(\nu_1, \gamma_2 + 1)) = \operatorname{rank}(H^{A,Y}(1, \gamma_2 + 1))$ for all $\nu_1 \geq 1$. ■

We have found bounds for γ_1 and γ_2 for the block Wiedemann and block Krylov sequences, $\{X^{\mathrm{T}} A^i Y\}_{i=0}^{\infty}$ and $\{A^i Y\}_{i=0}^{\infty}$, respectively, using general projection matrices X and Y. However, since these block projections will be randomly chosen in most cases,

we can also bound the probability of γ_1 and γ_2 being smaller. Yet again, we consider the rank of the block Hankel matrix.

By Villard (1997a, Corollary 1) and the Schwartz-Zippel Lemma, $\gamma_1 \leq \lceil n/\beta_1 \rceil$ and $\gamma_2 \leq \lceil n/\beta_2 \rceil$ for random matrices X and Y with $\beta_1 \geq \beta_2$. In other words, only the first $\lceil n/\beta_1 \rceil + \lceil n/\beta_2 \rceil$ entries of the block Wiedemann sequence are needed to compute the minimal generating matrix polynomial $F_X^{A,Y}$ for random projections X and Y.

The bounds $\gamma_1 \leq \lceil n/\beta_1 \rceil$ and $\gamma_2 \leq \lceil n/\beta_2 \rceil$ hold for random projection matrices X and Y, but it is possible to make bad random choices for the projections.

Example 4.2. Consider the companion matrix to the irreducible polynomial $\lambda^4 - 2$,*

$$A = \begin{bmatrix} 0 & 0 & 0 & 2 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \in \mathbb{R}^{4\times 4},$$

and the block projection matrices

$$X = Y = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}^{\mathrm{T}} \in \mathbb{R}^{4\times 2}.$$

The degree of the minimal generating matrix polynomial is $\deg(F_X^{A,Y}) = \gamma_2 = 4$. We can use the FPHPS algorithm from Theorem 4.10 on page 117 with $\nu_1 = \nu_2 = 4$ to

*Thank you to Gilles Villard for suggesting this example.

compute the minimal generating matrix polynomial $F_X^{A,Y}$. In this case, Theorem 4.10 gives the matrix

$$G = \begin{bmatrix} 1 - \frac{1}{2}\lambda^4 & 0 \\ 0 & 1 \end{bmatrix}$$

whose columns form a basis over $\mathbb{F}[\lambda]$ for $\mathbb{G}_X^{A,Y}$. Thus, the minimal generating matrix polynomial is

$$F_X^{A,Y} = \begin{bmatrix} 0 & \lambda^4 - 2 \\ 1 & 0 \end{bmatrix} = G \begin{bmatrix} 0 & -2 \\ 1 & 0 \end{bmatrix}.$$

The block projection vector Y has introduced zero vectors, forcing γ_2 and the degree of the minimal generating matrix polynomial to increase.

Example 4.3. Consider the matrices A and X given by Example 4.2 along with the block projection matrix

$$Y = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}^{\mathrm{T}} \in \mathbb{R}^{4\times 2}.$$

Here, $\gamma_2 = 2 = \lceil n/\beta_2 \rceil$. However, the first six matrices in the block Wiedemann sequence are

$$X^{\mathrm{T}}Y = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix},\ X^{\mathrm{T}}AY = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix},\ X^{\mathrm{T}}A^2Y = \begin{bmatrix} 0 & 2 \\ 0 & 0 \end{bmatrix},$$

$$X^{\mathrm{T}}A^3Y = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}, \; X^{\mathrm{T}}A^4Y = \begin{bmatrix} 2 & 0 \\ 0 & 0 \end{bmatrix}, \text{ and } X^{\mathrm{T}}A^5Y = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}.$$

This means $\operatorname{rank}(H_X^{A,Y}(3,3)) = 3$, but $\operatorname{rank}(H_X^{A,Y}(4,3)) = 4$, which is maximal. Thus, $\gamma_1 = 4 > 2 = \lceil n/\beta_1 \rceil$. The block projection vector X has introduced zero vectors, forcing γ_1 to increase. $F_X^{A,Y}$ cannot be computed from only the first $\lceil n/\beta_1 \rceil + \lceil n/\beta_2 \rceil = 4$ entries of the block Wiedemann sequence $\{X^{\mathrm{T}}A^iY\}_{i=0}^{\infty}$. Instead, we need $\{X^{\mathrm{T}}A^iY\}_{i=0}^{5}$. This time, we can use the FPHPS algorithm from Theorem 4.10 on page 117 with $\nu_1 = 4$ and $\nu_2 = 2$ to compute the minimal generating matrix polynomial $F_X^{A,Y}$. In this case, Theorem 4.10 gives the matrix

$$G = \begin{bmatrix} 1 & -\lambda^2 \\ -\frac{1}{2}\lambda^2 & 1 \end{bmatrix}$$

whose columns form a basis over $\mathbb{F}[\lambda]$ for $\mathbb{G}_X^{A,Y}$. Thus, the minimal generating matrix polynomial is

$$F_X^{A,Y} = \begin{bmatrix} \lambda^2 & -2 \\ -1 & \lambda^2 \end{bmatrix} = G \begin{bmatrix} 0 & -2 \\ -1 & 0 \end{bmatrix}.$$

Example 4.4. Finally, let $X = Y$ where the matrices A and Y are given by Example 4.3.

Again, $\gamma_2 = 2 = \lceil n/\beta_2 \rceil$. In this case, the first four matrices in the block Wiedemann

sequence are

$$X^{\mathrm{T}}Y = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \; X^{\mathrm{T}}AY = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix},$$
$$X^{\mathrm{T}}A^2Y = \begin{bmatrix} 0 & 2 \\ 1 & 0 \end{bmatrix}, \quad \text{and } X^{\mathrm{T}}A^3Y = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}.$$

This means $\mathrm{rank}(H_X^{A,Y}(2,3)) = 4$, which is maximal. Thus, $\gamma_1 = 2 = \lceil n/\beta_1 \rceil$, and Theorem 4.10 on page 117 with $\nu_1 = \nu_2 = 2$ once again returns the matrix

$$G = \begin{bmatrix} 1 & -\lambda^2 \\ -\frac{1}{2}\lambda^2 & 1 \end{bmatrix}$$

whose columns form a basis over $\mathbb{F}[\lambda]$ for $\mathbb{G}_X^{A,Y}$. Thus, the minimal generating matrix polynomial is again

$$F_X^{A,Y} = \begin{bmatrix} \lambda^2 & -2 \\ -1 & \lambda^2 \end{bmatrix} = G \begin{bmatrix} 0 & -2 \\ -1 & 0 \end{bmatrix}.$$

4.4 Block Rank Algorithm

The Kaltofen-Saunders rank algorithm discussed in Section 1.1.3 is a Monte Carlo algorithm to compute the rank of a matrix A from the minimal polynomial $f^{\tilde{A}}$ of the preconditioned matrix $\tilde{A}$. The algorithm relies on preconditioning the matrix A so that

the minimal polynomial $f^{\tilde{A}}$ of the preconditioned matrix $\tilde{A}$ is, with high probability, $f^{\tilde{A}} = \lambda\, g(\lambda)$ where $\text{rank}(A) = \deg(g)$ and $g(0) \neq 0$. The algorithm then returns

$$\text{rank}(A) = \deg(f^{\tilde{A}}) - 1 = \deg(f^{\tilde{A}}) - \text{codeg}(f^{\tilde{A}})$$

where the co-degree, $\text{codeg}(g)$, of the polynomial g is the degree of the smallest term with nonzero coefficient. In other words,

$$g(\lambda) = \sum_{i=\text{codeg}(g)}^{\deg(g)} g[i]\lambda^i$$

where $g[\text{codeg}(g)] \neq 0$ and $g[\deg(g)] \neq 0$.

To convert this algorithm to a block version, we must first know how the invariant factors of the minimal generating matrix polynomial $F_X^{A,Y}$ of the block Wiedemann sequence $\{X^{\mathrm{T}} A^i Y\}_{i=0}^{\infty}$ relate to the invariant factors of the characteristic matrix $\lambda I - A$. If $\beta_1 \geq \beta_2$, the i-largest invariant factor of $F_X^{A,Y}$ divides the i-largest invariant factor of $\lambda I - A$ (Kaltofen and Villard, 2001, Theorem 1). We want to know the probability they are equal for random block projections X and Y. This probability can be found by examining determinantal degree of $F_X^{A,Y}$ and the rank of the block Hankel matrix $H_X^{A,Y}(\nu_1, \nu_2 + 1)$ for random block projections. Lemma 4.6 on page 96 gives the rank of the block Hankel matrix $H_X^{A,Y}(\nu_1, \nu_2 + 1)$ as a lower bound for the determinantal degree of the minimal generating matrix polynomial $F_X^{A,Y}$. The divisibility of the i-th largest invariant factor of the characteristic matrix $\lambda I - A$ by the i-th largest invariant factor of

$F_X^{A,Y}$ given in Theorem 1 of Kaltofen and Villard (2001) gives an upper bound. Thus, the determinantal degree $\deg(\det(F_X^{A,Y}))$ of $F_X^{A,Y}$ is trapped between these two quantities, and the $\deg(\det(F_X^{A,Y}))$ can be found when these quantities are equal. Lemma 4.13 gives the probability of this happening for random block projections X and Y.

Lemma 4.13. *Let $\mathbb{F}$ be a field, S be a finite subset of $\mathbb{F}$, $A \in \mathbb{F}^{n\times n}$, and $1 \le \beta_2 \le \beta_1 \le n$. Let N be the sum of the degrees of the β_2 largest invariant factors of the characteristic matrix $\lambda I - A$,*

$$N = \sum_{i=0}^{\beta_2-1} \deg(s_{n-i}(\lambda I - A)).$$

If $X \in S^{n\times\beta_1}$ and $Y \in S^{n\times\beta_2}$ are matrices whose entries are chosen uniformly and independently from S, then the rank of the block Hankel matrix

$$H_X^{A,Y}(\nu_1, \nu_2+1) = \begin{bmatrix} X^TY & X^TAY & \cdots & X^TA^{\nu_2}Y \\ X^TAY & X^TA^2Y & \cdots & X^TA^{\nu_2+1}Y \\ \vdots & \vdots & \ddots & \vdots \\ X^TA^{\nu_1-1}Y & X^TA^{\nu_1}Y & \cdots & X^TA^{\nu_1+\nu_2-1}Y \end{bmatrix}$$

where $\nu_1 = \lceil N/\beta_1 \rceil$ and $\nu_2 = \lceil N/\beta_2 \rceil$ is

$$\operatorname{rank}(H_X^{A,Y}(\nu_1, \nu_2+1)) = N,$$

with probability at least $1 - 2N/|S| \ge 1 - 2n/|S|$.

Proof. The proof follows from Villard (1997a, Corollary 1) and the Schwartz-Zippel

Lemma.

Let $\mathcal{X}$ and $\mathcal{Y}$ be matrices whose entries are indeterminates $\zeta_{i,j}$ and $\xi_{i,k}$, respectively, over $\mathbb{F}$ where $1 \leq i \leq n$, $1 \leq j \leq \beta_1$, and $1 \leq k \leq \beta_2$. The lemma is true for a submatrix of the block Hankel matrix

$$H_{\mathcal{X}}^{A,\mathcal{Y}}(\nu_1, \nu_2+1) = \begin{bmatrix} \mathcal{X}^{\mathrm{T}}\mathcal{Y} & \mathcal{X}^{\mathrm{T}}A\mathcal{Y} & \cdots & \mathcal{X}^{\mathrm{T}}A^{\nu_2}\mathcal{Y} \\ \mathcal{X}^{\mathrm{T}}A\mathcal{Y} & \mathcal{X}^{\mathrm{T}}A^2\mathcal{Y} & \cdots & \mathcal{X}^{\mathrm{T}}A^{\nu_2+1}\mathcal{Y} \\ \vdots & \vdots & \ddots & \vdots \\ \mathcal{X}^{\mathrm{T}}A^{\nu_1-1}\mathcal{Y} & \mathcal{X}^{\mathrm{T}}A^{\nu_1}\mathcal{Y} & \cdots & \mathcal{X}^{\mathrm{T}}A^{\nu_1+\nu_2-1}\mathcal{Y} \end{bmatrix}$$

(Villard, 1997a, Corollary 1; 1997b, Corollary 6.4). Thus, there exists an $N \times N$ minor of $H_{\mathcal{X}}^{A,\mathcal{Y}}(\nu_1, \nu_2+1)$ that is not identically zero, and all larger minors are zero. Let the nonzero $N \times N$ minor of $H_{\mathcal{X}}^{A,\mathcal{Y}}(\nu_1, \nu_2+1)$ be denoted $\det\left(\left(H_{\mathcal{X}}^{A,\mathcal{Y}}(\nu_1, \nu_2+1)\right)^{[i_1,\ldots,i_N;j_1,\ldots,j_N]}\right)$. The entries of the matrix $H_{\mathcal{X}}^{A,\mathcal{Y}}(\nu_1, \nu_2+1)$ are polynomials in the indeterminates $\zeta_{i,j}$ and $\xi_{i,k}$ of degree at most two, so the minor $\det\left(\left(H_{\mathcal{X}}^{A,\mathcal{Y}}(\nu_1, \nu_2+1)\right)^{[i_1,\ldots,i_N;j_1,\ldots,j_N]}\right)$ is a polynomial in the indeterminates of degree at most $2N$. If X and Y are the matrices that result from choosing values for $\zeta_{i,j}$ and $\xi_{i,k}$ uniformly and independently from S, the minor $\det\left(\left(H_{X}^{A,Y}(\nu_1, \nu_2+1)\right)^{[i_1,\ldots,i_N;j_1,\ldots,j_N]}\right)$ is a nonzero element of $\mathbb{F}$ with probability at least $1 - 2N/|S|$ by the Schwartz-Zippel Lemma. In addition, because all larger minors of $H_{\mathcal{X}}^{A,\mathcal{Y}}(\nu_1, \nu_2+1)$ are zero, the larger minors of $H_{X}^{A,Y}(\nu_1, \nu_2+1)$ must also be zero. Thus, the rank of $H_{X}^{A,Y}(\nu_1, \nu_2+1)$ must be N. ■

We are now able to present the probability that the i-th largest invariant factor

of the minimal generating matrix polynomial $F_X^{A,Y}$ of the block Wiedemann sequence $\{X^{\mathrm{T}}A^iY\}_{i=0}^{\infty}$ is the i-th largest invariant factor of the characteristic matrix $\lambda I - A$. This is essentially a proof of the second half of Theorem 1 of Kaltofen and Villard (2001).

Theorem 4.13. *Let $\mathbb{F}$ be a field, S be a finite subset of $\mathbb{F}$, $A \in \mathbb{F}^{n\times n}$, and $1 \le \beta_2 \le \beta_1 \le n$. Let N be the sum of the degrees of the β_2 largest invariant factors of the characteristic matrix $\lambda I - A$,*

$$N = \sum_{i=0}^{\beta_2-1} \deg(s_{n-i}\left(\lambda I - A\right)).$$

If $X \in S^{n\times\beta_1}$ and $Y \in S^{n\times\beta_2}$ are matrices whose entries are chosen uniformly and independently from S, then $\gamma_1 \le \lceil N/\beta_1 \rceil \le \lceil n/\beta_1 \rceil$, $\gamma_2 \le \lceil N/\beta_2 \rceil \le \lceil n/\beta_2 \rceil$, and the i-th largest invariant factor of the minimal generating matrix polynomial $F_X^{A,Y}$ of the block Wiedemann sequence $\{X^TA^iY\}_{i=0}^{\infty}$ is the i-th largest invariant factor of the characteristic matrix $\lambda I - A$,

$$s_{\beta_2-i}\left(F_X^{A,Y}\right) = s_{n-i}\left(\lambda I - A\right), \qquad 0 \le i \le \beta_2 - 1,$$

with probability at least $1 - 2N/\left|S\right| \ge 1 - 2n/\left|S\right|$.

Proof. By Theorem 4.11 on page 126 and Lemma 4.6 on page 96,

$$\operatorname{rank}(H_X^{A,Y}(\nu_1, \nu_2 + 1)) \le \deg(\det(F_X^{A,Y})) \le N$$

for all $\nu_1 \geq 0$ and $\nu_2 \geq 1$. However, if $\nu_1 = \lceil N/\beta_1 \rceil$ and $\nu_2 = \lceil N/\beta_2 \rceil$, then

$$\operatorname{rank}(H_X^{A,Y}(\nu_1, \nu_2 + 1)) = N$$

with probability at least $1 - 2N/|S| \geq 1 - 2n/|S|$ by Lemma 4.13 on page 135. ■

The probability of the invariant factors being equal given by Theorem 4.13, along with the rank preconditioner presented in Section 3.2, gives a block Monte Carlo method to compute the rank of a singular matrix.

Lemma 4.14. *Let $\mathbb{F}$ be a field, S be a finite subset of $\mathbb{F}$, $A \in \mathbb{F}^{n\times n}$ have rank r with $r \leq n-1$ and a nonzero $r \times r$ principal minor, and let $1 \leq \beta_2 \leq \beta_1 \leq n$. Let N be the sum of the degrees of the β_2 largest invariant factors of the characteristic matrix $\lambda I - A$,*

$$N = \sum_{i=0}^{\beta_2 - 1} \deg(s_{n-i}(\lambda I - A)).$$

Let $X \in S^{n\times\beta_1}$ and $Y \in S^{n\times\beta_2}$ be matrices whose entries are chosen uniformly and independently from S, and let $D = \operatorname{diag}(d_1, \ldots, d_n)$ where $d_1, \ldots, d_n$ are chosen uniformly and independently from S. Then, the rank of A is

$$r = \deg(\det(F_X^{AD,Y})) - \operatorname{codeg}(\det(F_X^{AD,Y}))$$

and $F_X^{AD,Y}$ can be computed from the first $\lceil N/\beta_1 \rceil + \lceil N/\beta_2 \rceil$ matrices in the block Wiedemann sequence $\{X^T A^i Y\}_{i=0}^{\infty}$ with probability at least $1 - (4N + r(r+1))/(2|S|) \geq$

$1 - n(n+3)/(2\,|S|)$.

Proof. By Theorem 3.4 on page 60, the characteristic polynomial of the preconditioned matrix AD is $\det(\lambda I - AD) = \lambda^{n-r} g(\lambda)$ where $g(0) \neq 0$ and the minimal polynomial of AD is $f^{AD} = \lambda\ g(\lambda)$ and has degree $\deg(f^{AD}) = r+1$, all with probability at least $1 - r(r+1)/(2\,|S|)$. Thus, the product of the β_2 largest invariant factors of $\lambda I - AD$ is

$$\prod_{i=0}^{\beta_2-1} s_{n-i}\,(\lambda I - AD) = \lambda^k g(\lambda)$$

where $1 \leq k \leq n-r$, $g(0) \neq 0$, and $\deg(g) = r$ with probability at least $1 - r(r+1)/(2\,|S|)$.

At the same time,

$$\det(F_X^{AD,Y}) = \prod_{i=0}^{\beta_2-1} s_{\beta_2-i}\left(F_X^{AD,Y}\right) = \prod_{i=0}^{\beta_2-1} s_{n-i}\,(\lambda I - AD)$$

and $F_X^{AD,Y}$ can be computed from the first $\lceil N/\beta_1 \rceil + \lceil N/\beta_2 \rceil$ matrices in the block Wiedemann sequence $\{X^{\mathrm{T}} A^i Y\}_{i=0}^{\infty}$ with probability at least $1 - 2N/\,|S| \geq 1 - 2n/\,|S|$ by Theorem 4.13 on page 137. Thus, the rank r of A is

$$r = \deg(g) = \deg(\det(F_X^{AD,Y})) - \operatorname{codeg}(\det(F_X^{AD,Y}))$$

with probability at least

$$\left(1 - \frac{2N}{|S|}\right)\left(1 - \frac{r(r+1)}{2\,|S|}\right) \geq 1 - \frac{4N + r(r+1)}{2\,|S|} \geq 1 - \frac{n(n+3)}{2\,|S|}.$$

■

A similar argument will hold for DA and $F_X^{DA,Y}$.

Lemma 4.14 gives a block Monte Carlo approach to computing the rank of a singular matrix A. Along with Theorem 4.2 in Chen *et al.* (2002), it gives a block Monte Carlo approach to computing the rank of any matrix A.

Theorem 4.14. *Let $\mathbb{F}$ be a field, S be a finite subset of $\mathbb{F} \setminus \{0\}$, $A \in \mathbb{F}^{n\times n}$ have rank r and a nonzero $r \times r$ principal minor, and let $1 \le \beta_2 \le \beta_1 \le n$. Let N be the sum of the degrees of the β_2 largest invariant factors of the characteristic matrix $\lambda I - A$,*

$$N = \sum_{i=0}^{\beta_2 - 1} \deg(s_{n-i}(\lambda I - A)).$$

Let $X \in S^{n\times\beta_1}$ and $Y \in S^{n\times\beta_2}$ be matrices whose entries are chosen uniformly and independently from S, and let $D = \operatorname{diag}(d_1, \ldots, d_n)$ where $d_1, \ldots, d_n$ are chosen uniformly and independently from S. Then, the rank of A is

$$r = \deg(\det(F_X^{AD,Y})) - \operatorname{codeg}(\det(F_X^{AD,Y}))$$

and $F_X^{AD,Y}$ can be computed from the first $\lceil N/\beta_1 \rceil + \lceil N/\beta_2 \rceil$ matrices in the block Wiedemann sequence $\{X^T A^i Y\}_{i=0}^{\infty}$ with probability at least $1 - n(n+3)/(2\,|S|)$.

Proof. When A is singular, the proof follows directly from Lemma 4.14 on page 138. Otherwise, if A is nonsingular, $\det(AD) \neq 0$ and $r = n$. Furthermore, the characteristic and minimal polynomials of the preconditioned matrix AD are equal, $\det(\lambda I - AD) = f^{AD}$, with probability at least $1 - n(n-1)/(2\,|S|)$ by Chen *et al.* (2002, Theorem 4.2).

This means $\operatorname{codeg}(f^{AD}) = 0$ and the rank of A is

$$r = \deg(f^{AD}) = \deg(f^{AD}) - \operatorname{codeg}(f^{AD})$$

with probability at least $1 - n(n-1)/(2\,|S|)$.

At the same time,

$$\det(F_X^{AD,Y}) = \prod_{i=0}^{\beta_2-1} s_{\beta_2-i}\left(F_X^{AD,Y}\right) = \prod_{i=0}^{\beta_2-1} s_{n-i}\left(\lambda I - AD\right) = f^{AD}$$

and $F_X^{AD,Y}$ can be computed from the first $\lceil N/\beta_1 \rceil + \lceil N/\beta_2 \rceil$ matrices in the block Wiedemann sequence $\{X^{\mathrm{T}} A^i Y\}_{i=0}^{\infty}$ with probability at least $1 - 2N/\,|S| \geq 1 - 2n/\,|S|$ by Theorem 4.13 on page 137. Thus, the rank r of A is

$$r = \deg(\det(F_X^{AD,Y})) - \operatorname{codeg}(\det(F_X^{AD,Y}))$$

with probability at least

$$\begin{aligned}\left(1 - \frac{2N}{|S|}\right)\left(1 - \frac{n(n-1)}{2\,|S|}\right) &\geq 1 - \frac{4N + n(n-1)}{2\,|S|} \geq 1 - \frac{4n + n(n-1)}{2\,|S|} \\ &\geq 1 - \frac{n(n+3)}{2\,|S|}.\end{aligned}$$

■

Again, a similar argument will hold for DA and $F_X^{DA,Y}$.

We now have a complete block Monte Carlo algorithm to compute the rank of any matrix A. First, precondition the matrix A so the preconditioned matrix $\tilde{A}$ has a nonzero

$r \times r$ principal minor. For example, pre- and post-multiply by butterfly network preconditioners. Then, construct the minimal generating matrix polynomial $F_X^{\tilde{A}D,Y}$ and compute the rank of A from Theorem 4.14 on page 140.

Input: $A \in \mathbb{F}^{n \times n}$, S a finite subset of $\mathbb{F} \setminus \{0\}$, and $1 \leq \beta_2 \leq \beta_1 \leq n$

Output: $r = \text{rank}(A)$

1: $B_1, B_2 \leftarrow$ butterfly network preconditioners with parameters chosen uniformly and independently from S

2: $D \leftarrow \text{diag}(d_1, \ldots, d_n)$, $d_1, \ldots, d_n$ chosen uniformly and independently from S

3: $\tilde{A} \leftarrow {B_1}^{\mathrm{T}} A B_2 D$

4: Choose $X \in S^{n \times \beta_1}$ and $Y \in S^{n \times \beta_2}$ uniformly and independently

5: $\nu_1 \leftarrow \lceil n/\beta_1 \rceil$ and $\nu_2 \leftarrow \lceil n/\beta_2 \rceil$

6: Compute $F_X^{\tilde{A},Y}$ from $\{X^{\mathrm{T}} A^i Y\}_{i=0}^{\nu_1+\nu_2-1}$ (E.g., via the FPHPS algorithm.)

7: $r \leftarrow \deg(\det(F_X^{\tilde{A},Y})) - \text{codeg}(\det(F_X^{\tilde{A},Y}))$

This is a Monte Carlo method to compute the rank of a matrix A. There is currently no certificate for the rank of a matrix over an arbitrary field. However, this Monte Carlo method will always return a value that is no larger than the rank of the matrix.

Theorem 4.15. *Let $\mathbb{F}$ be a field, $A \in \mathbb{F}^{n \times n}$ have rank r, and $1 \leq \beta_2 \leq \beta_1 \leq n$. Let $X \in \mathbb{F}^{n \times \beta_1}$, $Y \in \mathbb{F}^{n \times \beta_2}$, and $D = \text{diag}(d_1, \ldots, d_n) \in \mathbb{F}^{n \times n}$. Then, the rank of A is bounded*

from below by

$$\begin{aligned} r &\geq \deg(\det(\lambda I - AD)) - \operatorname{codeg}(\det(\lambda I - AD)) \\ &\geq \deg(\det(F_X^{AD,Y})) - \operatorname{codeg}(\det(F_X^{AD,Y})). \end{aligned}$$

Proof. By Theorem 3.5 on page 63, λ^{n-r} divides the characteristic polynomial $\det(\lambda I - AD)$ of AD. Thus, $\operatorname{codeg}(\det(\lambda I - AD)) \geq n - r$ and

$$\deg(\det(\lambda I - AD)) - \operatorname{codeg}(\det(\lambda I - AD)) \leq n - (n - r) = r.$$

The i-th largest invariant factor of $F_X^{AD,Y}$ divides the i-th largest invariant factor of the characteristic matrix $\lambda I - AD$ (Kaltofen and Villard, 2001, Theorem 1; 2002), so $\det(F_X^{AD,Y})$ divides the characteristic polynomial $\det(\lambda I - AD)$. Let

$$\det(F_X^{AD,Y}) = \lambda^{k_1} g_1(\lambda) \quad \text{and} \quad \det(\lambda I - AD) = \lambda^{k_2} g_2(\lambda),$$

where $g_1(0) \neq 0$ and $g_2(0) \neq 0$. Then, $g_1 \mid g_2$, $\deg(g_2) \geq \deg(g_1)$,

$$\deg(\det(\lambda I - AD)) - \operatorname{codeg}(\det(\lambda I - AD)) = \deg(g_2),$$

and

$$\deg(\det(F_X^{AD,Y})) - \operatorname{codeg}(\det(F_X^{AD,Y})) = \deg(g_1). \qquad \blacksquare$$

Theorem 4.15 means that the block Monte Carlo rank algorithm will always return a

value that is no larger than the rank of the original matrix A.

This algorithm is similar to the one presented by Kaltofen and Saunders (1991, Theorem 3). In this block version, if the butterfly network preconditioners B_1 and B_2 are constructed using the generic exchange matrix of Chen *et al.* (2002, Section 6.2), then they each use at most $n\lceil\log_2(n)\rceil/2$ random elements from S (Chen *et al.*, 2002, Theorem 6.2) and are PreCondInd preconditioners with probability at least $1-r\lceil\log_2(n)\rceil/|S|$ (Chen *et al.*, 2002, Theorem 6.3). Thus, the leading $r \times r$ principal minor of ${B_1}^{\mathrm{T}} A B_2$ is nonzero with probability at least

$$\left(1 - \frac{r\lceil\log_2(n)\rceil}{|S|}\right)^2 \geq 1 - \frac{2r\lceil\log_2(n)\rceil}{|S|} \geq 1 - \frac{2n\lceil\log_2(n)\rceil}{|S|}$$

(Chen *et al.*, 2002, Theorem 3.1). Thus, the complete algorithm uses a total of at most

$$2\frac{n\lceil\log_2(n)\rceil}{2} + n + \beta_1 n + \beta_2 n = n\left(\beta_1 + \beta_2 + 1 + \lceil\log_2(n)\rceil\right)$$

random elements from S and returns the correct rank with probability at least

$$\begin{aligned}\left(1 - \frac{2r\lceil\log_2(n)\rceil}{|S|}\right)\left(1 - \frac{n(n+3)}{2|S|}\right) &\geq 1 - \frac{n(n+3) + 4r\lceil\log_2(n)\rceil}{2|S|} \\ &\geq 1 - \frac{n(n+3+4\lceil\log_2(n)\rceil)}{2|S|}.\end{aligned}$$

For comparison, the probability that the minimal polynomial $f_u^{A,v}$ of the Wiedemann sequence $\{u^{\mathrm{T}}A^i v\}_{i=0}^{\infty}$ is equal to the minimal polynomial $f^{\tilde{A}}$ of the matrix A is at least $1 - 2\deg(f^A)/|S|$ (Kaltofen and Pan, 1991, Lemma 2), which is the probability given

by Theorem 4.13 on page 137 with blocking factors $\beta_1 = \beta_2 = 1$. Thus, the Kaltofen-Saunders rank algorithm with the same preconditioner uses no more than

$$2\frac{n\lceil \log_2(n)\rceil}{2} + 3n = n\left(3 + \lceil \log_2(n)\rceil\right)$$

random elements from S and returns the correct rank with probability at least

$$1 - \frac{4\deg(f^{\tilde{A}}) + r(r+1) + 4r\lceil \log_2(n)\rceil)}{2\,|S|} \geq 1 - \frac{n(n+3+4\lceil \log_2(n)\rceil)}{2\,|S|}.$$

The block rank algorithm with a diagonal preconditioner and blocking factors $\beta_1 = \beta_2 = 1$ is the original Kaltofen-Saunders rank algorithm with the same preconditioner and that increasing the blocking factors causes an increase in the number of random field elements required and a decrease in the algorithms probability of success.

On the other hand, the block algorithm has two advantages over the non-blocked form. One is that it is a parallel algorithm (Coppersmith, 1994; Kaltofen, 1995; Villard, 2000). The other is that it captures more than just the largest invariant factor of the characteristic matrix $\lambda I - A$. The rank preconditioner used in Theorem 4.14 on page 140 does not take advantage of this, but other preconditioners may exist that do.

Chapter 5

LinBox Design

The dominant goal of the LinBox project is to produce algorithms and software for symbolic linear algebra, in particular black box linear algebra. The LinBox library is "middleware", focusing primarily on the algorithms and not implementing the underlying field arithmetic or the final interface a user might encounter. It is designed to be plugged into another package such as Maple or Mathematica. On the other end of the spectrum, the library uses generic or reusable programming to access existing libraries to implement the field arithmetic needed in the library's algorithms, allowing the code to operate over many coefficient domains and a variety of implementations for any given domain (Dumas *et al.*, 2002).

At the top level, the library provide algorithms for many standard problems in linear algebra. As input, these algorithms accept black box matrices. Any object conforming with the specification for a black box matrix can be plugged into these algorithms.

At a lower level, our code must operate over many coefficient domains. A user must even be able to easily change the implementation of a given coefficient domain. For instance, one might plug any of several implementations of the integers modulo a prime number into our code. We might want to use Victor Shoup's Number Theory Library (NTL) class `ZZ_p`, which implements the field using arbitrary length integers and residue arithmetic [`shoup.net`], or we might want to use an implementation that performs the field operations through Zech logarithm tables. We might also plug in a field of rational functions. We might or even use floating point numbers, although the resulting methods may not be numerically stable. We can capture many future improvements on field arithmetic without rewriting our programs. At very, stage we have applied the principle of generic or reusable programming.

Project LinBox implements generic programming through C++ templates and virtual member functions. Using the template mechanism, we are able to write our code only once (Stroustrup, 1997, Chapters 12 and 13). By providing the coefficient domain over which it acts as a template parameter, the code may be instantiated for many different domains. For example, the function `foo` is template-parameterized by the domain over which it acts:

```
template <class Domain>
Domain foo(Domain a, Domain b) { return a + b; }
```

The function returns the sum of its two inputs, and it may be applied to any type **Domain** that has the required `+` operator. One can then call the function `foo(x,y)`,

where `x` and `y` are the C++ types `double`, and the function `foo(m,n)` where `m` and `n` are the C++ types `int`. In both cases, the function `foo` is automatically instantiated for the type on which it acts. We may also call the instantiation explicitly using the function calls `foo<double>(x,y)` and `foo<int>(m,n)`.

The LinBox library follows Java's naming convention for all classes and functions internal to the library. This means names composed of multiple words are combined with capital letters for each word. Also, the first letter of class names is capitalized, while it is not for function names. For example, `MyClass` and `myFunction` are valid names for a LinBox class and function, respectively.

5.1 Archetypes

The LinBox library provides archetype classes for fields and black box matrices. An archetype serves three purposes: to define the common object interface, to supply one instance of the library as distributable compiled code, and to control code bloat.

An archetype is an abstract class whose role is similar to that of a Java interface. It specifies exactly what methods and members an explicitly designed class must have to be a pluggable template parameter type. A programmer can use the archetype as a template to create new fields or black box matrices that satisfy the interface. One does not need to resort to printed documentation about the library.

One disadvantage of the C++ template mechanism is that the source code for template functions and classes must be available to the programmer. Templatized libraries

cannot be distributed as compiled code because compilation requires a concrete instance for each templatized object.

An archetype provides a way to overcome this disadvantage and conceal the library source code from a programmer. Through the use of pointers and virtual member functions, the field archetype can hook into different LinBox fields. The templatized code may be compiled and distributed using the archetype as the template parameter. One can then access the template functions through the archetype mechanism.

Another disadvantage of the template mechanism is that compilation of a template-parameterized function or class causes compiled code to be created for each instantiation of the template. By compiling the code using the archetype as the template parameter, less compiled code is created.

Sections 5.3 and 5.5 discuss the archetypes for LinBox fields and black box matrices, respectively, in greater detail.

5.2 Integers

The LinBox library needs to be able to use matrices and vectors of very large dimensions. To do this, the library has a type `integer` to index matrix and vector elements and for any other function requiring an integer type. The LinBox type `integer` is a widening of the C++ `long` type. Any type that implements arbitrarily long integers and has all the operations of a C++ `long` may be used. This means, among other things, that the operations `+`, `-`, `*`, `/`, and `%` must be defined for the type, along with their inplace coun-

terparts: `+=`, `-=`, `*=`, `/=`, and `%=`. The type must also have default and copy constructors. LinBox uses GNU multiprecision integers for the type `integer` by default.

5.3 Fields, Elements, and Random Generators

The algorithms in the LinBox library are designed to operate with a variety of domains. In particular, we are interested in the finite fields of integers modulo a prime number. To perform the required arithmetic for these finite fields, algorithms must have access to additional parameters. For the fields of integers modulo a prime number, the parameter required is the prime modulus. To allow the code to be generic, the prime modulus cannot be passed directly to the algorithms. Other fields may have different parameters. Some fields, such as the field of rational numbers, have no parameters. Our code must operate on all possible fields.

Instead of passing these parameters explicitly to the algorithms, we want to incorporate them into the objects that represent the field and its elements. There are several ways to this. One way would be to explicitly store the required parameters in each field element object, but this would require a copy of the parameters for each element. One could store the parameters once and instead include pointers to them from each field element, but this requires memory to store the pointers. With both of these options, the parameters could easily be changed for each field element. The parameters would not need to be the same for all elements. It is possible to reduce the storage requirement by storing the parameters as global variables. This is the approach taken in Victor Shoup's

Number Theory Library (NTL), for example. There is no added storage for the parameters with each element, but it is then impossible to operate over more than one field at the same time.

The LinBox library takes a different approach. Our design uses two separate objects: a field object and an element object, where the element type is encapsulated within the field type:

```
Field F;

Field::Element x;
```

Here, **Field** is an user-defined or library type.

The element object contains only basic information about the element representation and is oblivious to what field it belongs. It may be of type C++ `long`, for integers modulo a word-size prime, or a more complicated data structure declared elsewhere in the library. The field interface requires only that the data type support C++ assignment and a copy constructor. (See Appendix B.1.)

The field object contains all of the methods requiring access to the field parameters. A LinBox field type contains constructors, destructors, and assignment and equality operators for field objects. (See Appendix B.2.)

Because elements by themselves do not have access to the field parameters, a LinBox field contains an initialization method. Every LinBox element must be initialized by a field before it is are used. For example, the function call `F.init(x,5)` initializes the element `x` to have a value in the field `F` corresponding to the integer 5. What this

corresponding value is depends on the field being used. This initialization is especially important because pointers may be used in the field element implementation. In this case, initialization ensures the pointers point to a valid object.

LinBox fields provide additional methods that may be used once the element objects have been initialized. For instance, all fields contain methods for assignment and equality-testing of element objects. The function call `F.areEqual(x,y)` tests whether the elements `x` and `y` are equal in the field `F` and returns a boolean `true` or `false`. Similarly, the function calls `F.isZero(x)` and `F.isOne(x)` test whether the element `x` is the zero or one of the field `F`.

LinBox fields also provide methods for arithmetic such as addition and subtraction of element objects similar to the `+` and `-` operators for C++ types. For example, the function call `F.add(x,y,z)` adds the elements `y` and `z` in the field `F`, stores the result in the element `x`, and returns a reference to `x`. Similarly, a LinBox field contains the methods `F.sub(x,y,z)`, `F.mul(x,y,z)`, and `F.div(x,y,z)` for subtraction, multiplication, and division. Thus, the code

```
Field F;
Field::Element x, y, z;
F.init(y,2);
F.init(z,3);
F.mul(x,y,z);
```

can be used to multiply 2 times 3 in the field `F`. A LinBox field also contains two methods

to compute the additive and multiplicative inverses of a field element. The function call `F.neg(x,y)` stores the additive inverse (negation) of `y` in the field `F` in the element `x`. Similarly, `F.inv(x,y)` makes `x` the multiplicative inverse of `y` in the field `F`.

In addition to these arithmetic methods, LinBox fields contain methods for inplace arithmetic such as is done by the C++ operators `+=` and `-=`. For example, `F.subin(x,y)` subtracts the element `y` from `x` in the field `F` and stores the result in the element `x`. Similarly, the methods `F.addin(x,y)`, `F.mulin(x,y)`, `F.divin(x,y)`, `F.negin(x)`, and `F.invin(x)` are provided for inplace addition, multiplication, division, negation, and multiplicative inversion.

Because addition and multiplication and are often performed together, for example in computing the inner product of two vectors, the function call `F.axpy(z,a,x,y)` multiplies `a` with `x` and adds the product to `y` in the field `F`. The result is stored in the element `z` and returned as a reference. Similarly, `F.axpyin(y,a,x)` is the inplace version of the method, storing the result in the element `y`. These methods may result in some performance gain. For example, in a field of integers modulo a prime number, the modular reduction may be delayed until after both the addition and multiplication are performed.

Finally, LinBox fields contain methods for printing and reading field and element objects to and from an IO stream such as `cin` and `cout`. Thus, the function call `F.write(cout,x)` writes the element `x` to the output stream `cout`, and `F.read(cin,x)` reads the element `x` from the input stream `cin`.

Any LinBox field must contain these basic methods; more methods may be created by

using derived classes. Template-parameterized algorithms and objects may also be used to add additional functionality. For example, the `FieldAXPY` object wraps the `axpyin` method of a field.* It is template-parameterized by a LinBox field and stores the result of the `axpyin` operation so that the `FieldAXPY` object contains the running sum of the products. It is constructed from a field object, and it assigns the initial value of zero to the sum. Thus, the code

```
FieldAXPY Y(F);

Y.accumulate(a,d);

Y.accumulate(b,e);

Y.accumulate(c,f);

y = Y.get();
```

computes the value `y = a*d + b*e + c*f`. The function call `Y.assign(x)` may also be used to assign the value of the field element `x` to the stored value of `Y`. The `FieldAXPY` object may be used to compute the inner product of two vectors, and template specializations allow a programmer to gain performance. For instance, the modular reduction may be delayed even further in a field of integers modulo a prime.

For each field type **Field**, there exists an encapsulated class **Field**`::RandIter` that uniformly generates random elements of the field or an unspecified subset of a given cardinality. (See Appendix B.3.) Many of the algorithms in the LinBox library depend on the availability of such random field elements. (See, for example, the Wiedemann

*Much of the work of the current `FieldAXPY` was implemented by Bradford Hovinen.

nonsingular system solver discussed in Section 1.1.1 and the Kaltofen-Saunders rank algorithm discussed in Section 1.1.3.)

Every field, whether or not it requires parameters such as a prime modulus, to perform arithmetic, has the same interface. Because there is no distinction between parametric and unparametric fields, and because a default constructor does not make sense for a parametric field, we cannot require a LinBox field to have a default constructor. This means LinBox algorithms cannot employ a default constructor for an abstract field. The algorithms can, however, use a copy constructor. Every field must be created explicitly and passed to the algorithm.

The LinBox library provides a template wrapper class for wrapping a C++ data type into an unparametric field meeting the LinBox interface. For example, the LinBox field `UnparametricField<double> F` wraps the C++ `double` type into an object `F` meeting the interface of a LinBox field. The LinBox field `F` acts on elements of the type `UnparametricField<double>::Element`, which is exactly the type `double`. The wrapper can be used on any type or class that supports the standard assignment, arithmetic, and equality operations: `=`, `+`, `-`, `*`, `==`, etc. If a type implements a method in a different manner, one can use a partial template specialization to define the corresponding field method. For example, the class `ZZ_p` of Victor Shoup's NTL implements most of the standard operations, and the following example adjusts the `inv` function to the signature of LinBox's `inv` method.

```
template <> NTL::ZZ_p&
UnparametricField<NTL::ZZ_p>::inv(
```

```
    NTL::ZZ_p& x, const NTL::ZZ_p& y
) const
{ return x = NTL::inv(y); }
```

This allows us to easily adapt fields from other libraries to LinBox. The added level of indirection created by the template wrapper class is relatively small because the compiler will remove most of it during compilation (Stroustrup, 1997, Section 2.7.1). Thus, we are able to use the compiler to ease the job of the programmer.

The performance lost by wrapping a field with this wrapper varies from field to field and even among the methods of a given field. For example, Table 5.1 on the following page shows the time needed to compute a given number of additions and multiplications using `NTL::zz_p`. This test, which was inspired by Dumas *et al.* (2002, Section 2), initializes a field element to the zero value and then runs a loop the indicated number of times. This loop first adds $a = 3$ to the value and then multiplies it by $b = 1100$. The test was run on a dual-processor 750 MHz Pentium III machine with 1 GB of memory running Red Hat 7.1. The test was compiled with the GNU Project's open-source `g++-2.96` C++ compiler distributed with Red Hat 7.1 using the `-O3` optimization flag, which turns on all optimization except unrolling loops and strict aliasing. In particular, all simple functions are integrated into their callers.

The first two rows of the table show the time used when calling `NTL::zz_p` directly and as the LinBox field `UnparametricField<NTL::zz_p>`. The third row of the table shows the corresponding time when elements are initialized at each step. We see there is

Table 5.1: Time in seconds to compute the N additions and multiplications using `NTL::zz_p` directly and as the LinBox field `UnparametricField<NTL::zz_p>` using the prime 32749.

N	10^7	2×10^7	3×10^7	4×10^7	5×10^7	6×10^7
Directly	1.38	2.75	4.15	5.51	6.92	8.34
LinBox field	1.34	2.73	4.07	5.39	6.79	8.14
LinBox field with `init`	9.83	19.60	29.42	39.20	49.02	59.05

no increase in running time when using the field wrapper if when no new field elements are initialized, but the initialization is very costly.

A user can also supply a new field to LinBox code. This user-supplied field may perform better than a wrapped field. For example, the LinBox library contains the field `Modular` of integers modulo a prime number. This field is template-parameterized by an integer type representation for the field elements. Table 5.2 shows the time needed to compute the same number of additions and multiplications using the LinBox field `Modular<long>`. We see the running time is less than that of even using `NTL::zz_p` directly, partly because the `NTL::zz_p` has additional overhead to allow word-sized primes to be used. `Modular<long>` does not ensure the arithmetic operations do not overflow the memory allocated to the representation. This makes this field faster, but it also limits the size of the prime the field may use.

Table 5.2: Time in seconds to compute the N additions and multiplications using `Modular<long>` using the prime 32749.

N	10^7	2×10^7	3×10^7	4×10^7	5×10^7	6×10^7
LinBox field	1.25	2.50	3.77	5.06	6.39	7.57

The field archetype defines the interface that all field implementations must satisfy. Any class that meets that interface can be hooked into a generic algorithm. The archetype `FieldArchetype` points to an abstract base class `FieldAbstract` that introduces a virtual copy method ("clone"). (See Figure 5.1.) This added level of indirection allows the archetype to have an STL-compliant copy constructor, thus allowing the library to use the default allocator in the Standard Template Library.

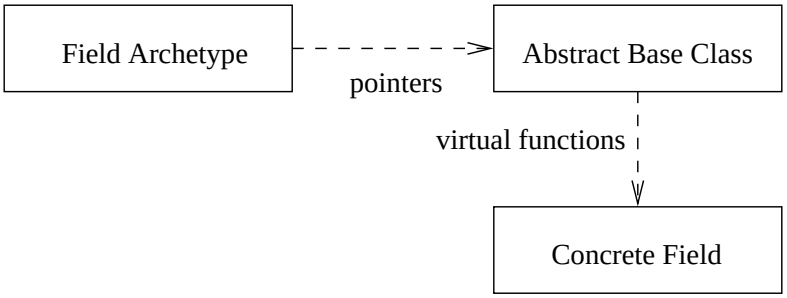

Figure 5.1: LinBox field archetype

To aid in hooking any archetype-compliant field type onto this abstract class, LinBox provides a template class called `FieldEnvelope`. For example, we saw earlier that the field `UnparametricField<double> F` complies with the LinBox field interface. Then, the envelope object `FieldEnvelope< UnparametricField<double> > E(F)` is of a class derived from the abstract field type `FieldAbstract` and can be used by the field archetype. A LinBox field archetype object may be constructed from a pointer to any field that complies with the LinBox field interface by way of a `FieldEnvelope` object. Thus, the field archetype object `FieldArchetype A(&F)` contains a pointer to the `FieldEnvelope` ob-

ject created by wrapping the field `F` in the `FieldEnvelope` template class.[†]

A generic algorithm can also be instantiated with the archetype and compiled separately. Code making use of this algorithm can supply a field inherited from the abstract field type and link this field against the compiled code. This will cause a modest performance loss resulting from the inability to inline field operations and from the additional memory indirection.

This performance loss can be seen by comparing the running times for library code using both the LinBox field itself and the field archetype. For example, Table 5.3 on the next page shows the running time in seconds to compute the minimal polynomial f^A of an $n \times n$ Trefethen matrix for several dimensions n using Wiedemann's method and three LinBox fields of integers modulo a prime number. (See Section 5.6 for a description of the Trefethen matrix.) As in the previous test, this test is run using the LinBox field `Modular<long>` and the wrapped field `UnparametricField<NTL::zz_p>` of `NTL::zz_p` elements. As before, the test was run on a dual-processor 750 MHz Pentium III machine with 1 GB of memory running Red Hat 7.1 and was compiled with the `g++-2.96` C++ compiler using the `-O3` optimization flag.

Table 5.4 on the following page shows the corresponding running times using the field archetype. The code using the field archetype takes longer than the code without the archetype. In this example, we see the additional running time in using the field archetype is approximately the same for both fields and a given dimension n. This is

[†]The `FieldEnvelope` template class and the `FieldArchetype A(&F)` constructor were suggested and implemented by Jean-Guillaume Dumas.

Table 5.3: Time in seconds to compute the minimal polynomial of a $n \times n$ Trefethen matrix using the LinBox fields `Field1=Modular<long>` and `Field2=UnparametricField<NTL::zz_p>` of integers modulo the prime 32749.

n	1000	2000	3000	4000	5000	6000	7000	8000
`Field1`	0.83	3.51	8.14	14.73	23.43	34.07	46.88	61.61
`Field2`	1.19	4.89	11.26	20.47	32.08	46.63	63.87	83.85

to be expected since the additional time results from dereferencing pointers and the number of pointers to be dereferenced should depend on the size of the problem and not the field being used. As before, the wrapped NTL field has longer running times than `Modular<long>` because of the overhead involved in ensuring the representation will not overflow its storage, but this is somewhat balanced by the limitations on the prime that can be used.

Table 5.4: Time in seconds to compute the minimal polynomial of a $n \times n$ Trefethen matrix using `FieldArchetype` and the LinBox fields `Field1=Modular<long>` and `Field2=UnparametricField<NTL::zz_p>` of integers modulo the prime 32749.

n	1000	2000	3000	4000	5000	6000	7000	8000
`Field1`	2.73	11.32	25.90	60.07	117.1	160.23	266.68	360.75
`Field2`	3.02	15.23	29.42	54.15	101.29	191.50	276.27	395.42

5.4 Vectors

LinBox has both dense and sparse vectors. Dense vectors store every entry of the vector, making no distinction between zero and nonzero field elements. To implement dense

vectors in the LinBox library, we decided to incorporate the C++ Standard Template Library (STL) vector of field elements. However, we disallow any vector methods that would invalidate any iterators. This means a LinBox algorithm is not allowed to perform any operation that may change the size of a dense vector that is passed as an argument to the code. These operations include the assignment operator `=` and the vector methods `assign`, `push_back`, `pop_back`, `insert`, `erase`, `resize` and `reserve` (Musser and Saini, 1996, Section 6.1).

Sparse vectors, on the other hand, store only the nonzero entries of the vector. There are two types of LinBox sparse vectors: sparse sequence vectors and sparse associative vectors. Sparse sequence vectors are implemented like an STL sequence of pairs of the LinBox integer indices of the nonzero elements and the corresponding field elements. For example, an STL list `std::list< std::pair< integer, Element > >` or an STL vector `std::vector< std::pair< integer, Element > >`. This implementation is efficient for iterating through the nonzero entries. However, given an index, finding the vector entry must be implemented as a linear search of the STL sequence container.

Sparse associative vectors provide a better implementation to access a vector entry by index. They have the interface of an STL unique and associative container of the LinBox integer index values and the corresponding field elements. In other words, they may be implemented via an STL map `std::map<integer, Element>`. However, the C++ map operator `operator[]` is disallowed to avoid storing zero values in the sparse associative vector. Due to its data structure, a map entry access is logarithmic time (Musser and

Saini, 1996, Section 7.2).

5.4.1 Traits

To aid in writing code for the various vector types, the LinBox library uses the trait technique introduced by Myers (1995). Vector traits can include data members to be used as tags for the type of vector and aid code in distinguishing which algorithm should be applied to a given vector.

To accomplish this, the library contains a structure `VectorCategories` that contains tags for each of the three vector types:

```
struct VectorCategories
{
  template <class T> struct DenseVectorTag
    { typedef T Trait; };
  template <class T> struct SparseSequenceVectorTag
    { typedef T Trait; };
  template <class T> struct SparseAssociativeVectorTag
    { typedef T Trait; };
};
```

To allow easy access to the trait class, the tag structures are template-parameterized by the trait class `T`.[‡] Such access can be useful to further specialize code that has already

[‡]Thank you to Дмитрий Морозов (Dmitriy Morozov) for suggesting and implementing the template parameterization in the vector trait classes.

been specialized by the vector category tag.

The vector trait class is template-parameterized by the vector class to which it refers and contains two types. The type `VectorType` is the vector to which the trait class refers. The other, `VectorCategory`, is the vector category tag type. For example, the specialization for the STL vector of field elements is the following:

```
template <class Element>
struct VectorTraits< std::vector<Element> >
{
  typedef std::vector<Element> VectorType;
  typedef typename
    VectorCategories::DenseVectorTag<
        VectorTraits<VectorType>
    > VectorCategory;
};
```

In addition to this basic tagging feature, traits can be used to provide additional functionality for the vectors. For example, the trait classes for the sparse sequence vector types might provide a `sort` function to sort the pairs of indices and field elements in ascending ordering of the indices. This sorting must be implemented differently for random access sequence containers such as the STL vector than for forward iterator containers such as the STL list. The trait class allows the library to provide a uniform calling mechanism.

5.4.2 Subvectors

It may be useful at times to pass only part of a vector to a LinBox function. For example, a block algorithm, such as discussed in Chapter 4, may store matrices internal to the algorithm as an STL vector of field elements. The rows and columns of the matrices can then be thought of as subvectors of this LinBox dense vector.

At access these subvectors, the LinBox library provides a subvector wrapper class `Subvector<Vector>` that wraps any dense vector type. A `Subvector<Vector>` object is constructed from a dense vector and three integers.

```
Subvector(v, start, stride, length) s;
```

One integer, `start`, determines where the subvector starts in relation to the parent vector. Another, `length`, designates the length of the subvector. The third, `stride`, indicates how far apart the entries of the subvector are on the parent vector. Thus, a $m \times n$ matrix A is stored as a dense vector v such that the (i,j)-th entry of A is the $(m(i-1)+j)$-th entry of v, $A^{[i,j]} = v^{[m(i-1)+j]}$, the i-th row of A can be accessed with `start` $= im$, `length` $= n$, and `stride` $= 1$; and the j-th column of A can be accessed with `start` $= j$, `length` $= m$, and `stride` $= n$.

Example 5.1. The matrix

$$A = \begin{bmatrix} 3 & 1 & 4 & 6 \\ 3 & 2 & 1 & 3 \end{bmatrix}$$

can be stored as the dense vector

$$v = \begin{bmatrix} 3 & 1 & 4 & 6 & 3 & 2 & 1 & 3 \end{bmatrix}^{\mathrm{T}}.$$

The i-th row of A can be accessed as a subvector of v with `start` $= 2i$, `length` $= 4$, and `stride` $= 1$. Similarly, the j-th column of A can be accessed as a subvector of v with `start` $= j$, `length` $= 2$, and `stride` $= 4$. Indexing always starts at zero, not one.

5.5 Black Box Matrices

The LinBox black box matrix archetype is simpler than the field archetype because the design constraints are less stringent. As with the field type, we need a common object interface to describe how algorithms are to access black box matrices, but it only requires functions to access the matrix's dimensions and to apply the matrix or its transpose to a vector. (See Appendix B.4.) In particular, the black box matrix archetype does not need to have a copy constructor like the field archetype does. Instead, the function call `B.clone()` returns a pointer to a new black box matrix archetype object that is a copy of the black box matrix archetype object `B`. Thus, our black box matrix archetype is simply an abstract class. (See Figure 5.2 on the next page.) In addition, the overhead involved with the inheritance and virtual methods is negligible in comparison with the execution time of the methods. This was not the case for the field and element types. This relatively small overhead allows us to use the black box archetype directly in LinBox

library algorithms. Thus, we do not template-parameterize our algorithms with a black box type. The user provides a specific derived class of the black box archetype to the algorithm.

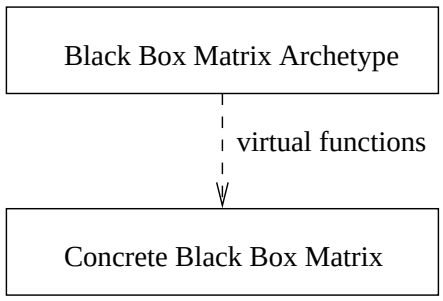

Figure 5.2: LinBox black box matrix archetype

The black box matrix archetype contains methods to access the matrix's dimensions. The function calls `B.coldim()` and `B.rowdim()` return integers representing the column and row dimensions, respectively, of the matrix `B`. These are not true dimensions, but rather an upper bound on the dimensions of the nonzero entries of the matrix.

The LinBox black box matrix archetype also contains methods to apply the matrix to a vector. For example, the function call `B.apply(x,y)` computes the matrix-vector product of the matrix `B` and the vector `y`, stores the result in the vector `x`, and returns a reference to `x`. Similarly, the function call `B.applyTranspose(x,y)` computes the matrix-vector product of the transpose of the matrix `B` and the vector `y`, stores the result in the vector `x`, and returns a reference to `x`. The function calls `B.apply(x)` and `B.applyTranspose(x)` return references to the respective matrix-vector products without storing the result, and the function calls `B.applyIn(x)` and `B.applyTransposeIn(x)`

are inplace methods to place the results of the matrix-vector products in the same memory as the input vectors. For convenience, some of these methods have default implementations in the archetype. For example, `apply` and `applyTranspose` each have three variants that handle allocation of the input and output vectors differently. Only the first variant of each, `B.apply(x,y)` and `B.applyTranspose(x,y)`, must be implemented by a derived black box matrix class. The archetype base class provides default implementations of the other variants, but a derived class can override them.

The black box matrix archetype is template-parameterized by the vector type upon which the `apply` methods act, but not by the field in which the arithmetic is done. We saw no reason to template-parameterize the matrix archetype by the field since the derived matrix class may still incorporate the arithmetic field without the archetype base class needing to access it. This may even be implemented as a template parameter, but such genericity may not be required. The LinBox library contains several black box matrix classes that are not templatized by a field type. For example, the black box matrix type `Permutation` creates a permutation matrix that permutes the values of an input vector. The library also contains black box matrices `Transpose` to transpose a black box matrix and `Compose` to compose two matrices. In other words, if the black box matrix `C` is the composition of the black box matrices `A` and `B`, the function call `C.applyin(x)` yields the same result as first calling the function `B.applyin(x)` and then `A.applyin(x)`.

The LinBox library also contains several black box matrix types that are template-parameterized by a field type. The class `Diagonal` contains a diagonal black box matrix,

and the class `Hilbert` contains a Hilbert matrix. The library also contains two sparse matrix types, `SparseMatrix0` and `SparseMatrix1`. These sparse matrix classes both store each row of the matrix as a LinBox sparse vector.

The field is also always available as an argument to black box algorithms, which may perform additional coefficient field operations and need access to the field themselves. In addition, variants of the `apply` and `applyTranspose` methods are provided through which a user could pass additional information, including the field over which to operate. This might be useful in an algorithm such as the Kaltofen's baby steps/giant steps determinant algorithm for a dense integer matrix where several fields are used with the same matrix.

As discussed in Section 5.4, LinBox algorithms cannot invalidate iterators to the vector arguments. This means that the black box matrix apply methods for dense vectors cannot perform any operations that may change the size of the vectors passed as arguments to the method. It is the responsibility of the library user to ensure the vector arguments are of the correct size.

5.6 Trefethen's Challenge

Recently, Nick Trefethen issued a challenge to numerical analysts to solve ten problems, each to ten digits of accuracy (Trefethen, 2002). The problems, all numerically difficult, ranged across numerical analysis. One problem, number seven, is the computation of the $(1,1)$ entry of the inverse A^{-1} of the 20000×20000 matrix A whose entries are all zero except for the first 20,000 primes, $2, 3, 5, 7, \ldots, 224737$, along the main diagonal and the

number 1 on the diagonals whose distance from the main diagonal is a power of two. In other words, the number 1 is in all positions (i, j) such that $|i - j| = 2^k$ for $0 \leq k \leq 14$,

$$A^{[i,j]} = 1 \quad \text{if } |i - j| = 1, 2, 4, 8, \ldots, 16384.$$

The LinBox project took this as a challenge to compute the exact solution to the problem, which is a rational number with approximately 100,000 digits in each the numerator and denominator. This computation is far beyond the capability of Maple, Mathematica, and any other commercial computer algebra software system using current processors and memories (Dumas *et al.*, 2002, Section 4). Project members responded by using not one, but three different approaches. All three methods, however, attempted to solve the problem by finding the solution to the linear equation

$$Ax = e_1 = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \end{bmatrix}^{\mathrm{T}}.$$

The first entry of x is the $(1, 1)$ entry of A^{-1}.

Jean-Guillaume Dumas attacked the problem by using Cramer's rule, which says that the i-th component of the solution to the linear equation $Ax = b$ is the quotient of the determinant of the matrix formed by replacing the i-th column of A with the vector b and the determinant of the original matrix A,

$$x^{[i]} = \frac{\det(A \underset{i}{\leftarrow} b)}{\det(A)} = \frac{\det(A \underset{i}{\leftarrow} e_1)}{\det(A)}$$

(Horn and Johnson, 1985, Section 0.8.3). Thus, the solution to the problem can be found by taking the quotient of two determinants,

$$(A^{-1})^{[1,1]} = \frac{\det(A \underset{1}{\leftarrow} e_1)}{\det(A)}. \tag{5.1}$$

Because the only nonzero entry of the unit vector e_1 is the one in the first position, minor expansion says the numerator is the trailing $19,999 \times 19,999$ principal minor $\det\left(A^{[2,\ldots,20000;2,\ldots,20000]}\right)$ of A resulting from deleting the first row and column. Each determinant can be bounded either directly by Hadamard's inequality as in Section 1.1.5 or by using Geršgorin discs to bound the eigenvalues (Horn and Johnson, 1985, Section 6.1). The computation then involves computing the two determinants modulo multiple primes using the method described in Section 1.1.4. Finally, Chinese remaindering is used to construct the integer determinants. The total computation took approximately four days on 186 processors for a total of about two years' computation time, or about 1.7 seconds per digit, and used 11,946 primes for the numerator and 12,863 for the denominator.

We took a different approach. Instead of finding the numerator and denominator separately, we used one word-sized prime p and Hensel lifting instead of Chinese remaindering in a method first proposed by Moenck and Carter (1979) and then by Dixon (1982). The idea is to compute a solution modulo a large enough modulus q and recover the answer with continued fractions. The modulus q can be found by using Hadamard's inequality and Cramer's rule (5.1) to find a bound for the solution.

The method begins by initializing the vectors

$$b_0 = b = e_1 \quad \text{and} \quad \overline{x}_0 = x_0 = A^{-1}b_0 \bmod p.$$

Then, for $j \geq 1$,

$$b_j = \frac{b_0 - A\overline{x}_{j-1}}{p^j}, \quad x_j = A^{-1}b_j \bmod p, \quad \text{and} \quad \overline{x}_j = \overline{x}_{j-1} + p^j x_j.$$

Here, $\overline{x}_j \equiv x \pmod{p^{j+1}}$, so we are really trying to compute the first entry of $\overline{x}_{k-1}$ for k such that $p^{k-1} < q \leq p^k$. However,

$$\begin{aligned} b_0 - A\overline{x}_{j-1} &= b_0 - A\overline{x}_{j-2} - p^{j-1}x_{j-1} \\ &= p^{j-1}\left(b_{j-1} - x_{j-1}\right), \end{aligned}$$

so the update for b_j becomes

$$b_j = \frac{b_{j-1} - Ax_{j-1}}{p},$$

and only the first entry of $\overline{x}_j$ must be stored.

The algorithm requires two matrix-vector products. One, Ax_{j-1}, must be done exactly, but it is easily done with a black box matrix since A is very sparse. The other $A^{-1}b_j$ need only be done modulo p, but involves a dense matrix. We can use Wiedemann's nonsingular linear system solver presented in Section 1.1.1 to compute x_j without storing the entire dense matrix $A^{-1} \bmod p$. This modification, which was first seen in Kaltofen

and Saunders (1991, Section 3), may cause an increased running time but lower memory requirements since we can store the coefficients of the minimal polynomial $f^A \bmod p$ in less space than the inverse matrix. (See Appendix C for the entire algorithm.)

We did not run this computation to completion, but we observed that if we use a 30-bit prime, the minimal polynomial f^A is computed in approximately 15 minutes, and another 17 minutes is required for each Hensel lift. Thus, calculating the solution modulo q would take approximately 225 days on a single machine. It is possible to compute the solution modulo several relatively prime moduli q_i simultaneously on different processors and use Chinese remaindering to combine the result.

Finally, Zhendong Wan ran a version of Dixon's algorithm that used the explicit inverse of the matrix A. Using A^{-1} explicitly results in faster lifting steps than using the minimal polynomial to solve the system, but it requires much more memory. Using a machine with adequate memory resulted in computing the answer in twelve and a half days, or about 5.4 seconds per digit.

5.7 DOC++ Documentation

Documentation for a computer system usually comes in two forms. Programmers often document their code with comments in the source code itself. By keeping the documentation within the source code, the programmer finds keeping the documentation up to date with the source code much easier. However, this documentation is difficult to read, especially for a user who may not be familiar, or want to be familiar, with the actual

C++ code.

The user of the software often turns to a manual or some other source separate from the source code to understand the software's application program interface (API). This separation between documentation and code makes the documentation easier for the user to read, but it is more difficult to maintain for the same reason.

The LinBox library uses the DOC++ documentation system to document the library's application program interface (API). DOC++ documentation is inserted in the source code by the programmer using the same comment style as that of the better-known JavaDoc tool from Sun Microsystems. The DOC++ software can extract this information and create both HTML and LaTeX output that is easy for a user to read. By keeping the documentation within the source code, it is much easier for a programmer to keep the documentation up to date with the source code. At the same time, the JavaDoc comment style allows the programmer to include both comments to be used in the user's documentation and also comments that explain the details of the code to another programmer, which allows DOC++ to access only the information in which a user would be interested.

DOC++ documentation is hierarchically structured. This structure is reflected in the section structure of the LaTeX document and in the HTML page hierarchies. For both methods, DOC++ also generates an alphabetical index. DOC++ uses this hierarchical structuring to reflect the methods and members of a class, as well as the class derivation hierarchies. These class derivation hierarchies are also represented as graphs. Java applets

are used for graphs in the HTML output. The user can then click on a class to view its documentation.

Chapter 6

Summary and Future Work

The black box approach to linear algebra is known to both numerical analysis and symbolic computations, although the requirements of the two fields mean that methods in one may not be appropriate to the other. While numerical analysis uses floating point numbers to find approximations to the solution to an linear system, symbolic computation uses exact arithmetic and random values to find an exact but probabilistic answer.

The Wiedemann method in symbolic computation uses the minimal polynomial of the Krylov sequences and the bilinear projection sequences of a matrix. Using random projection vectors, these minimal polynomials can be used to solve many common problems in linear algebra. To do so, the matrix may need to be preconditioned through a combination of matrix pre- and post-multiplication.

Chapter 2 discussed preconditioners for the rank and singular system solving problems that are based on Beneš networks. Theorem 2.1 on page 25 generalized the butterfly

networks of Parker (1995) to arbitrary dimension n, and Theorem 2.2 on page 35 generalized these networks further to arbitrary radix switches. Sections 2.2 and 2.4 discussed how to obtain preconditioners to localize linear independence from these switching networks and generic exchange matrices. Further work should investigate how to implement generic exchange matrices for arbitrary radix switches more efficiently and thus improve the result of Theorem 2.3 on page 38.

Chapter 3 introduced a technique for using the determinantal divisors of a characteristic matrix to obtain preconditioners to ensure the cyclicity of the nonzero eigenvalues of the matrix. Theorem 3.2 on page 51, Corollary 3.1 on page 53, and Corollary 3.2 on page 54 used this technique to introduce a new determinant-preserving preconditioners for computing the determinant of a dense integer matrix through Kaltofen's baby steps/ giant steps algorithm. Then, Theorem 3.4 on page 60 slightly relaxed the condition of putting the matrix into a generic rank profile for the Kaltofen-Saunders rank algorithm. Future work needs to investigate preconditioners to take advantage of this relaxation.

Chapter 4 investigated the block Wiedemann approach, which uses blocks of vectors for projections, and the right minimal generating matrix polynomial for the block sequence. Section 4.1 discussed linearly generated matrix sequences and right generating matrix polynomials, and Definition 4.5 on page 78 defined the minimal generating matrix polynomial for a linearly generated matrix sequence. Section 4.2 discussed Beckermann and Labahn's Fast Power Hermite-Padé Solver (FPHPS) algorithm and how it can be used to compute the minimal generating matrix polynomial of a linearly generated matrix

sequence. Section 4.3 then considered the block Wiedemann and Krylov sequences for a matrix and showed how to compute the minimal generating matrix polynomial for the the two block sequences. Finally, Section 4.4 and Theorem 4.14 on page 140 introduced a block method for computing the rank of a matrix. Additional preconditioners that take advantage of the multiple invariant factors captured by the block Wiedemann method should also be investigated, as well as the relationship between various algorithms for computing the minimal generating matrix polynomial, including the FPHPS algorithm, Dickinson *et al.* and Coppersmith's block Berlekamp-Massey algorithms, and the original Berlekamp-Massey algorithm.

Finally, Chapter 5 discussed the design of the LinBox library, some issues involved in its creation and documentation, and generic programming with the C++ template mechanism. Section 5.1 introduced the concept of an archetype, and Section 5.2 examined the integer type used by the library. Sections 5.3, 5.4, and 5.5 presented the common object interfaces for fields, elements, vectors, and matrices. Section 5.4.1 examined the use of traits in LinBox, Section 5.6 discussed the exact solution of one of the problems proposed in Nick Trefethen's challenge, and Section 5.7 discussed the DOC++ documentation system used by the library. Work still needs to be done to implement additional fields and algorithms, including block Wiedemann methods and garbage-collected fields like those of SacLib.

References

L. Babai (1979). Monte Carlo Algorithms in Graph Isomorphism Testing. Technical Report DMS 79-10, Université de Montréal, Département de mathématiques et de statistique. [3]

Bernhard Beckermann and George Labahn (1994). A Uniform Approach for the Fast Computation of Matrix-Type Pade Approximants. *SIAM Journal on Matrix Analysis and Applications*, **15**(3): 804–823. [66, 99, 105, 107, 108, 111, 115, 176, 185]

V. E. Beneš (1964). Permutation Groups, Complexes, and Rearrangeable Connecting Networks. *Bell System Technical Journal*, **43**: 1641–1656. [20]

Elwyn R. Berlekamp (1968). *Algebraic Coding Theory*. McGraw-Hill, New York. [5]

R. B. Boppana and R. Hirschfeld (1989). Pseudorandom Generators and Complexity Classes. In S. Micali (editor), *Randomness and Computation*, volume 5 of *Advances in Computing Research*, pages 1–26. JAI Press, Inc., Greenwich, Conneticut. [3]

Li Chen, Wayne Eberly, Erich Kaltofen, B. David Saunders, William J. Turner, and Gilles Villard (2002). Efficient Matrix Preconditioners for Black Box Linear Algebra. *Linear Algebra and its Applications*, **343-344**: 119–146. Special issue on *Infinite Systems of Linear Equations Finitely Specified*, edited by P. Dewilde, V. Olshevsky and A. H. Sayed. [7, 11, 13, 19, 20, 29, 38, 41, 42, 43, 55, 59, 62, 63, 140, 144]

Don Coppersmith (1993). Solving Linear Equations over GF(2): Block Lanczos Algorithm. *Linear Algebra and its Applications*, **192**: 33–60. [15]

Don Coppersmith (1994). Solving Homogeneous Linear Equations Over GF(2) via Block Wiedemann Algorithm. *Mathematics of Compututation*, **62**: 333–350. [6, 15, 65, 145]

Angel Díaz (1997). *FoxBox: a System for Manipulating Symbolic Objects in Black Box Representation*. Ph.D. thesis, Rensselaer Polytechnic Institute, Troy, New York. [16, 17]

Angel Díaz and Erich Kaltofen (1998). FoxBox: a System for Manipulating Symbolic Objects in Black Box Representation. In Oliver Gloor (editor), *ISSAC'98: Proceedings of the 1998 International Symposium on Symbolic and Algebraic Computation*, pages 30–37. ACM Press, New York, New York. [17]

Bradley W. Dickinson, Martin Morf, and Thomas Kailath (1974). A Minimal Realization Algorithm for Matrix Sequences. *IEEE Transactions on Automatic Control*, **AC-19**(1): 31–38. [65, 66, 177]

John D. Dixon (1982). Exact Solution of Linear Equations Using P-Adic Expansions. *Numerische Mathematik*, **40**: 137–141. [170]

Jean Louis Dornstetter (1987). On the Equivalence Between Berlekamp's and Euclid's Algorithms. *IEEE Transactions on Information Theory*, **IT-33**(3): 428–431. [67]

J.-G. Dumas, T. Gautier, M. Giesbrecht, P. Giorgi, B. Hovinen, E. Kaltofen, B. D. Saunders, W. J. Turner, and G. Villard (2002). LinBox: A Generic Library for Exact Linear Algebra. In Arjeh M. Cohen, Xiao-Shan Gao, and Nobuki Takayama (editors), *International Congress of Mathematical Software*. World Scientific, Singapore. To appear, 10 pages. [17, 18, 146, 156, 169]

Jean-Guillaume Dumas (2000). Algorithmes parallèles efficaces pour le calcul formel: algèbre linéaire creuse et extensions algébriques. Thèse de Doctorat, Institut National Polytechnique de Grenoble, France, décembre 2000. [18]

Wayne Eberly and Erich Kaltofen (1997). On Randomized Lanczos Algorithms. In Küchlin (1997), pages 176–183. [7]

G. David Forney, Jr (1975). Minimal Bases of Rational Vector Spaces, with Applications to Multivariable Linear Systems. *SIAM Journal on Control*, **13**(3): 493–520. [81]

F. R. Gantmacher (1977). *Matrix Theory*, volume I. Chelsea Publishing Company. [58, 63, 129]

Joachim von zur Gathen and Jürgen Gerhard (1999). *Modern Computer Algebra*. Cambridge University Press, Cambridge. [3, 92, 102]

I. Gohberg, P. Lancaster, and L. Rodman (1982). *Matrix Polynomials*. Academic Press, New York. [46]

Gene H. Golub and Charles F. Van Loan (1989). *Matrix Computations*. The Johns Hopkins University Press, Baltimore, M.D., second edition. [1]

Magnus R. Hestenes and Eduard Stiefel (1952). Methods of Conjugate Gradients for Solving Linear Systems. *Journal of Research of the National Bureau of Standards*, **49**(6): 409–436. [1]

Roger A. Horn and Charles R. Johnson (1985). *Matrix Analysis*. Cambridge University Press, Cambridge. [14, 54, 170]

Thomas Kailath (1980). *Linear Systems*. Prentice Hall, Englewood Cliffs, NJ. [78]

E. Kaltofen and A. Lobo (1999). Distributed Matrix-Free Solution of Large Sparse Linear Systems over Finite Fields. *Algorithmica*, **24**(3–4): 331–348. Special issue on *Coarse Grained Parallel Algorithms*. [17]

Erich Kaltofen (1992a). Efficient Solution of Sparse Linear Systems. Lecture notes, Computer Science Department, Rensselaer Polytechnic Institute, Troy, N.Y. [67, 85]

Erich Kaltofen (1992b). On Computing Determinants of Matrices Without Divisions. In Paul S. Wang (editor), *ISSAC'92: Proceedings of the 1992 International Symposium on Symbolic and Algebraic Computation*, pages 342–349. ACM Press, New York, New York. [14]

Erich Kaltofen (1995). Analysis of Coppersmith's Block Wiedemann Algorithm for the Parallel Solution of Sparse Linear Systems. *Mathematics of Compututation*, **64**(210): 777–806. [65, 67, 145]

Erich Kaltofen (2000). Challenges of Symbolic Computation: My Favorite Open Problems. *Journal of Symbolic Computation*, **29**(6): 891–919. With an additional open problem by Robert M. Corless and David J. Jeffrey. [3, 4, 16]

Erich Kaltofen (2002). An Output-Sensitive Variant of the Baby Steps/Giant Steps Determinant Algorithm. In Teo Mora (editor), *ISSAC 2002: Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation*. ACM Press, New York, New York. To appear. [14]

Erich Kaltofen and Victor Pan (1991). Processor Efficient Parallel Solution of Linear Systems over an Abstract Field. In *Proceedings of SPAA '91 3rd Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 180–191. ACM Press, New York, New York. [12, 144]

Erich Kaltofen and Victor Pan (1992). Processor Efficient Parallel Solution of Linear Systems II: the Positive Characteristic and Singular Cases. In *Proceedings of the 33rd Annual Symposium on Foundations of Computer Science*, pages 714–723. IEEE Computer Society Press, Los Alamitos, California. [13, 15, 42, 43]

Erich Kaltofen and B. David Saunders (1991). On Wiedemann's Method of Solving Sparse Linear Systems. In H. F. Mattson, T. Mora, and T. R. N. Rao

(editors), *AAECC-9: Proceedings of the 1991 Applied Algebra, Algebraic Algorithms and Error-Correcting Codes, International Conference*, volume 539 of *Lecture Notes in Computer Science*, pages 29–38. Springer Verlag, Heidelberg, Germany. [5, 7, 8, 9, 10, 11, 13, 43, 58, 144, 171]

Erich Kaltofen and Barry M. Trager (1990). Computing with polynomials given by black boxes for their evaluations: Greatest common divisors, factorization, separation of numerators and denominators. *Journal of Symbolic Computation*, **9**(3): 301–320. [1]

Erich Kaltofen and Gilles Villard (2001). On the Complexity of Computing Determinants. In Kiyoshi Shirayanagi and Kazuhiro Yokoyama (editors), *Proceedings of the Fifth Asian Symposium on Computer Mathematics (ASCM 2001)*, volume 9 of *Lecture Notes Series on Computing*, pages 13–27. World Scientific, Singapore. Invited contribution; extended abstract. [13, 14, 15, 68, 127, 134, 135, 137, 143]

Erich Kaltofen and Gilles Villard (2002). Private communication. [14, 127, 143]

Wolfgang W. Küchlin (Editor) (1997). *ISSAC'97: Proceedings of the 1997 International Symposium on Symbolic and Algebraic Computation*. ACM Press, New York, New York. [179, 183]

B. A. LaMacchia and A. M. Odlyzko (1991). Solving Sparse Linear Systems over Finite Fields. In A. J. Menezes and S. Vanstone (editors), *Advances in Cryptology: CRYPTO'90*, volume 537 of *Lecture Notes in Computer Science*, pages 109–133. Springer Verlag, Heidelberg, Germany. [1]

Rob Lambert (1996). *Computational Aspects of Discrete Logarithms*. Ph.D. thesis, University of Waterloo, Waterloo, Ontario, Canada. [2, 15]

Cornelius Lanczos (1950). An Iteration Method for the Solution of the Eigenvalue Problem of Linear Differential and Integral Operators. *Journal of Research of the National Bureau of Standards*, **45**(4): 255–282. [1]

Cornelius Lanczos (1952). Solution of Systems of Linear Equations by Minimized Iterations. *Journal of Research of the National Bureau of Standards*, **49**(1): 33–53. [1]

Austin Lobo (1995). *Matrix-Free Linear System Solving and Applications to Symbolic Computation*. Ph.D. thesis, Rensselaer Polytechnic Institute, Troy, New York. [28]

James L. Massey (1969). Shift-Register Synthesis and BCH Decoding. *IEEE Transactions on Information Theory*, **IT-15**(1): 122–127. [5]

Robert T. Moenck and John H. Carter (1979). Approximate Algorithms to Derive Exact Solutions to Systems of Linear Equations. In Ng (1979), pages 65–73. [170]

Peter L. Montgomery (1995). A Block Lanczos Algorithm for Finding Dependencies Over GF(2). In Louis C. Guillou and Jean-Jacques Quisquater (editors), *Advances in Cryptology - EUROCRYPT '95, International Conference on the Theory and Application of Cryptographic Techniques*, volume 921 of *Lecture Notes in Computer Science*, pages 106–120. Springer Verlag, Heidelberg, Germany. [16]

David R. Musser and Atul Saini (1996). *STL Tutorial and Reference Guide: C++ Programming with the Standard Template Library.* Addison-Wesley, Reading, Massachusetts. [17, 161]

Nathan Myers (1995). A New And Useful Template Technique: "Traits". *Borland C++ Report.* [162]

Edward W. Ng (Editor) (1979). *Symbolic and Algebraic Computation, EUROSAM '79, An International Symposiumon Symbolic and Algebraic Computation, Marseille, France, June 1979, Proceedings*, volume 72 of *Lecture Notes in Computer Science.* Springer Verlag, Heidelberg, Germany. [181, 183]

D. Scott Parker (1995). Random Butterfly Transformations with Applications in Computational Linear Algebra. Technical Report CSD-950023, Computer Science Deptartment, University of California, Los Angeles. [20, 27, 29, 176]

V. M. Popov (1970). Some Properties of Control Systems with Irreducible Matrix Transfer Functions. In J. A. Yorke (editor), *Seminar on Differential Equations and Dynamical Systems, II*, volume 144 of *Lecture Notes in Computer Science*, pages 169–180. Springer Verlag, Heidelberg, Germany. [78]

V. M. Popov (1972). Invariant Description of Linear, Time-Invariant Controllable Systems. *SIAM Journal on Control*, **10**(2): 252–264. [77]

J. Rissanen (1971). Recursive Identification of Linear Systems. *SIAM Journal on Control*, **9**(3): 420–430. [66]

J. Rissanen (1972). Realization of Matrix Sequences. Technical report, IBM. [66]

B. D. Saunders (2001). Black Box Methods for Least Squares Problems. In Bernard Mourrain (editor), *ISSAC 2001: Proceedings of the 2001 International Symposium on Symbolic and Algebraic Computation*, pages 297–302. ACM Press, New York, New York. [29]

B. David Saunders, Arne Storjohann, and Gilles Villard (2001). Matrix Rank Certification. *Electronic Journal of Linear Algebra*, page to appear. [11]

J. T. Schwartz (1980). Fast Probabilistic Algorithms for Verification of Polynomial Identities. *Journal of the ACM*, **27**: 701–717. [50, 60]

Bjarne Stroustrup (1997). *The C++ Programming Language*. Addison-Wesley, Reading, Massachusetts, third edition. [17, 147, 156]

Nick Trefethen (2002). A Hundred-dollar, Hundred-digit Challenge. *SIAM News*, **35**(1). [168]

Marc Van Barel and Adhemar Bultheel (1991). The Computation of Non-Perfect Padé-Hermite Approximants. *Numerical Algorithms*, **1**: 285–304. [66, 105]

Marc Van Barel and Adhemar Bultheel (1992). A General Module Theoretic Framework for Vector and M-Padé and Matrix Rational Interpolation. *Numerical Algorithms*, **3**: 451–462. [66]

Gilles Villard (1997a). A Study of Coppersmith's Block Wiedemann Algorithm using Matrix Polynomials. In Küchlin (1997), pages 32–39. Extended abstract; proofs in Villard (1997b). [77, 81, 130, 135, 136]

Gilles Villard (1997b). A Study of Coppersmith's Block Wiedemann Algorithm using Matrix Polynomials. Rapport de Recherche 975 IM, Institut d'Informatique et de Mathématiques Appliquées de Grenoble. [136, 183]

Gilles Villard (1998). Block Solution of Sparse Linear Systems Over GF(q): the Singular Case. *SIGSAM Bulletin*, **32**(4): 10–12. [7]

Gilles Villard (2000). Processor Efficient Parallel Solution of Linear Systems of Equations. *Journal of Algorithms*, **35**(1): 122–126. [145]

Douglas H. Wiedemann (1986). Solving Sparse Linear Equations Over Finite Fields. *IEEE Transactions on Information Theory*, **IT-32**(1): 54–62. [2, 4, 5, 6, 8, 9, 10, 12, 13, 14, 20, 28, 42, 43, 44, 65]

Richard Zippel (1979). Probabilistic Algorithms for Sparse Polynomials. In Ng (1979), pages 216–226. [50, 60]

Richard Zippel (1990). Interpolating Polynomials from Their Values. *Journal of Symbolic Computation*, **9**(3): 375–403. [50, 60]

APPENDICES

Appendix A

Fast Power Hermite-Padé Solver

Input: $m \geq 2$, $s \in \mathbb{Z}_{\geq 0}$, $\mathscr{F} = (\mathscr{F}^{[1]}, \ldots, \mathscr{F}^{[m]})$, multiiindex $\mathscr{N} = (\mathscr{N}^{[1]}, \ldots, \mathscr{N}^{[m]})$ *

Output: $\{\mathscr{P}_{i,\sigma}\}_{i=1}^{m}$ and $\{\mathrm{dct}(\mathscr{P}_{i,\sigma}) = d_{i,\sigma} + 1\}_{i=1}^{m}$ such that any solution $\mathscr{P}$ to the power Hermite-Padé approximant problem can be written as $\mathscr{P} = \sum_{i=1}^{m} g_i \mathscr{P}_{i,\sigma}$ where $\deg(g_i) < \mathrm{dct}(\mathscr{P}_i)$.

1: **for** $1 \leq l \leq m$ **do**

2: $\quad d_{l,0} \leftarrow \mathscr{N}^{[l]}$

3: $\quad \mathscr{P}_{l,0} \leftarrow e_l$ /* *l-th unit vector* */

4: **end for**

5: **for** $0 \leq k \leq \sigma - 1$ **do**

6: $\quad$ **for** $1 \leq l \leq m$ **do**

7: $\quad\quad c_{l,k} \leftarrow \lambda^{-k} \mathscr{P}_{l,k}(\lambda^s) \cdot \mathscr{F}(\lambda)\big|_{\lambda=0}$

*This algorithm is the variation of the FPHPS algorithm described by Beckermann and Labahn (1994, Section 3) that is used in Section 4.2 and chooses π to have the smallest possible value whenever there is a choice in its value.

end for

$\Lambda_k \leftarrow \{l \mid c_{l,k} \neq 0\}$

if $\Lambda_k = \{\}$ **then**

  for $1 \leq l \leq m$ **do**

    $\mathscr{P}_{l,k+1} \leftarrow \mathscr{P}_{k,l}$

    $d_{l,k+1} \leftarrow d_{l,k}$

  end for

else

  $d_k \leftarrow \max\{d_{l,k} \mid l \in \Lambda_k\}$

  $\pi \leftarrow \pi_k \leftarrow \min\{j \mid d_{j,k} = d_k\}$

  for $1 \leq l \leq m$ **do**

    if $l \notin \Lambda_k$ **then**

      $\mathscr{P}_{l,k+1} \leftarrow \mathscr{P}_{l,k}$

      $d_{l,k+1} \leftarrow d_{l,k}$

    else if $l = \pi$ **then** /* $l \in \Lambda_k$ */

      $\mathscr{P}_{l,k+1} \leftarrow \lambda \mathscr{P}_{l,k}$

      $d_{l,k+1} \leftarrow d_{l,k} - 1$

    else /* $l \in \Lambda_k, l \neq \pi$ */

      $\mathscr{P}_{l,k+1} \leftarrow \mathscr{P}_{l,k} - \frac{c_{l,k}}{c_{\pi,k}} \mathscr{P}_{\pi,k}$

      $d_{l,k+1} \leftarrow d_{l,k}$

    end if

29: **end for**

30: **end if**

31: **end for**

Appendix B

LinBox Common Object Interface

B.1 Elements

```
Element(void); // Default constructor
Element(const Element& a);
  // Copy constructor
~Element(); // Destructor
Element& operator=(const Element& a);
  //Assignment
```

B.2 Fields

```
// Object Management

Field(const Field& F);
  // Copy Constructor
~Field();
  // Destructor
Field& operator=(const Field& F);
  // Assignment
Element& init(Element& x, const integer& y=0) const;
```

```
  // Element initialization from integer
integer& convert(integer& x, const Element& y) const;
  // Element conversion to integer
Element& assign(Element& x, const Element& y) const;
  // Assignment
integer& cardinality(integer& c) const;
  // Field cardinality
integer& characteristic(integer& c) const;
  // Field characteristic

// Arithmetic Operations

bool areEqual(const Element& x,
        const Element& y) const;
  // Equality
Element& add(Element& x,
                const Element& y,
                const Element& z) const;
  // Addition: x = y + z
Element& sub(Element& x,
                const Element& y,
                const Element& z) const;
  // Subtraction: x = y - z
Element& mul(Element& x,
                const Element& y,
                const Element& z) const;
  // Multiplication: x = y * z
Element& div(Element& x,
                const Element& y,
                const Element& z) const;
  // Division: x = y / z
Element& neg(Element& x, const Element& y) const;
  // Negation: x = - y
Element& inv(Element& x, const Element& y) const;
  // Multiplicative inverse: x = 1/y
Element& axpy(Element& r,
                const Element& a,
                const Element& x,
                const Element& y) const;
  // r = a*x + y

// Inplace Arithmetic Operations
```

```
bool isZero(const Element& x) const;
  // Addition: x = y + z
bool isOne(const Element& x) const;
Element& addin(Element& x, const Element& y) const;
  // Addition: x += y
Element& subin(Element& x, const Element& y) const;
  // Subtraction: x -= y
Element& mulin(Element& x, const Element& y) const;
  // Multiplication: x *= y
Element& divin(Element& x, const Element& y) const;
  // Division: x /= y
Element& negin(Element& x) const;
  // Negation: x = - x
Element& invin(Element& x) const;
  // Multiplicative inverse: x = 1/x
Element& axpyin(Element& r,
                const Element& a,
                const Element& x) const;
  // r += a*x

// Input/Output Operations

ostream& write(ostream& os) const; // Print field
istream& read(istream& is) const;  // Read field
ostream& write(ostream& os, const Element& x) const;
  // Print Element
istream& read(istream& is, Element& x) const;
  // Read Element
```

B.3 Random Element Generators

```
RandIter(const Field& F,
                const integer& size = 0,
                const integer& seed = 0);
  // Constructor from field, size of set, and seed
RandIter(const RandIter& R);
  // Copy constructor
~RandIter(); // Destructor
RandIter& operator=(const RandIter& R);
```

```
  // Assignment operator
Element& random(Element& x); // Random Element creator
```

B.4 Black Box Matrices

```
Blackbox_archetype* clone () const;
  // Virtual constructor.
Vector& apply (const Vector& x) const;
  // Application of black box matrix.  Not required.
Vector& apply (Vector& y, const Vector& x) const;
  // Application of black box matrix.
Vector &apply (Vector &y,
               const Vector &x,
               void *handle) const;
  // Application of black box matrix.
Vector& applyIn (Vector &x) const;
  // In-place application of black box matrix.
  // Not required.
Vector& applyTranspose (const Vector &x) const;
  // Application of black box matrix.  Not required.
Vector& applyTranspose (Vector& y, const Vector& x) const;
  // Application of black box matrix transpose.
Vector& applyTranspose (Vector& y,
               const Vector& x,
               void* handle) const;
  // Application of black box matrix transpose.
Vector& applyTransposeIn (Vector &x) const;
  // In-place application of black box matrix tranpose.
  // Not required.
integer rowdim (void) const;
  // Retreive row dimensions of black box matrix.
integer coldim (void) const;
  // Retreive column dimensions of black box matrix.
```

Appendix C

Algorithm for Trefethen's Challenge

Input: $n > 0$ and matrix $A \in \mathbb{Z}^{n \times n}$

Output: $x^{[1]} = (A^{-1})^{[1,1]}$

1: $q \leftarrow 2 \prod_{i=1}^{n} A^{[i]}$ where $A^{[i]}$ is the i-th column of A. /* *Hadamard's inequality* */

2: $k \leftarrow \lceil \log_p(k) \rceil$

3: $g(\lambda) = \sum_{i=0}^{d} g[i]\lambda^i \lambda^i \leftarrow f^A$

4: $g(\lambda) \leftarrow -g/g[0]$ /* *Normalize so* $g(0) = -1$ */

5: $b_0 \leftarrow e_1$

6: $x_0 \leftarrow \sum_{i=1}^{d} g[i] A^{i-1} b_0 \bmod p$

7: $\overline{x}_0{}^{[1]} \leftarrow x_0{}^{[1]}$

8: **for** $j = 1, 2, \ldots, k-1$ **do**

9: $\quad b_j \leftarrow (b_{j-1} - A x_{j-1})/p$

10: $\quad x_j \leftarrow \sum_{i=1}^{d} g[i] A^{i-1} b_j \bmod p$

11: $\overline{x}_j{}^{[1]} \leftarrow \overline{x}_{j-1}{}^{[1]} + p^j x_j{}^{[1]}$

12: **end for**

13: Recover $x^{[1]}$ from $x^{[1]} \equiv \overline{x}_{k-1}{}^{[1]} \bmod p^k$ by continued fractions