ABSTRACT

WADDELL, CLEVELAND ALEXANDER. Parametric Linear System Solving with Error Correction. (Under the direction of Dr. Erich Kaltofen).

Consider solving a black box linear system, $A(u)x = b(u)$, where the entries are polynomials in u over a field K, and $A(u)$ is full rank. The solution, $x = \frac{1}{g(u)}f(u)$, where g is always the least common monic denominator, can be recovered even if some evaluations are erroneous. In [Boyer and Kaltofen, Proc. SNC 2014] the problem is solved with an algorithm that generalizes Welch/Berlekamp decoding of an algebraic Reed-Solomon code. Their algorithm requires the sum of a degree bound for the numerators plus a degree bound for the denominator of the solution. We describe an algorithm that given the same inputs uses possibly fewer evaluations to compute the solution.

We introduce a second count for the number of evaluations required to recover the solution based on work by Stanley Cabay. The Cabay count includes bounds for the highest degree polynomial in the coefficient matrix and right side vector, but does not require solution degree bounds. Instead our algorithm iterates until the Cabay termination criterion is reached. At this point our algorithm returns the solution. Assuming we have the actual degrees for all necessary input parameters, we give the criterion that determines when the Cabay count is fewer than the generalized Welch/Berlekamp count.

We then specialize the algorithm for parametric linear system solving to the recovery of a vector of rational functions, $\frac{1}{g(u)}f(u)$. If the rational function vector is the solution to a full rank linear system our early termination strategy applies and we may recover it from fewer evaluations than generalized Welch/Berlekamp decoding. We then show that if entries in our rational function vector are polynomials, then the vector can be viewed as an interleaved Reed-Solomon code. Thus if the errors occur in bursts we can again do better than generalized Welch/Berlekamp decoding.

The aforementioned algorithms do not work when the matrix of the system, $A(u)x = b(u)$, is rank deficient and some evaluations cause errors. We next present an algorithm for solving black box linear systems where the entries are polynomials over a field and the matrix of the system is rank deficient. The algorithm first locates and removes all errors, after which it computes a solution that satisfies the input degree bounds.

Finally we view the recovery of a vector of polynomials as the decoding of interleaved Reed-Solomon codes. It is known that interleaved schemes improve the error correction capabilities of codes when errors occur in bursts. If we consider that the evaluations of the polynomials in the vector are done one at a time and that the errors are dependent then it is likely that errors occur in consecutive evaluations (bursts). An error model that considers consecutive entries in the vector to be wrong is a burst error model. We show how to encode and transmit data (polynomials) so that the data (polynomials) transmitted can be recovered with fewer evaluations than is required by Generalized Welch/Berlekamp decoding.

Parametric Linear System Solving with Error Correction

by
Cleveland Alexander Waddell

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

Raleigh, North Carolina

2019

APPROVED BY:

Dr. Terrence Blackman External Member	Dr. Ernie Stitzinger
Dr. Hoon Hong	Dr. Kailash Misra

Dr. Erich Kaltofen
Chair of Advisory Committee

DEDICATION

To my grandfather the late Rev. Wilford Alexander Morris, my first formal teacher, his sister Zephreen Moriah for takig such good care of me as an infant. And to mother Bridget Marilyn Morris my father Dr. Clairmont Alexis Waddell,

BIOGRAPHY

Cleveland Alexander Waddell, the last of my four children, the second of two boys, was born in 1987, September 28. His early education, Kindergarten - High School, was initiated and completed in Guyana, South America, At the Pre-Kindergarten level he was ready and enthusiastic about attending school, especially the one his cousin was already attending. Pre-High School days, Cleveland is known to have been very playful yet evidenced a capability to balance play with sporting, church-related activities, and academic prowess.

It might be apt to say that Mathematics took a hold of him for classmates at high school sought his help with key concepts, he tutored students in his community, and he was decisive in his desire to pursue, abroad, a university level education and consequent qualification. Medgar Evers College afforded him such baccalaureate opportunity and he firmly grasped the challenge and honed his skill and aptitude in the Computer Science and Mathematics.That unwavering focus on scoring in cricket and basketball, and striding ahead of competitors in track events was applied to Graduate studies at North Carolina State University. The many Awards represent the acknowledgement and recognition of his scholarship.

Cleveland is, presently, part of a group that is promoting proficiency in Mathematics during the Summertime in Guyana. In his view, Mathematics could be fun. If my son had tutored me, I would have done better in Chemistry and Mathematics!

This young man, his learning taken in hand at an early age by his grand-father, understands the need for recreation to help manage the demands of academic pursuits, and add to a rounded persona.The pages of his life are but a fraction written, the rest is open and waiting to be punctuated by variables, so to speak.

ACKNOWLEDGEMENTS

I wish to acknowledge the support by the National Science Foundation.

I would like to thank my adviser, Prof. Erich Kaltofen, for his problem, time and patience. This document would not be what it is without his guidance and support throughout the writing process. I would also like to express my gratitude to my committee members for sharing with me their wisdom and experience.

I would like to thank the academic support staff of the Department of Mahtematics during my tenure at the University for their help in ensuring all necessary administrative actions were taken in a timely manner. Your professionalism meant that I could focus my attention on finishing this document.

Graduate school can at times be a source of great stress. I am lucky to have the love and support of many family members, friends and well wishers. Some of whom have listened to me talk through problems even though they didn't quite follow. I am appreciative of the many conversations, board games, sporting events, baked goods, bike rides, dinners, gym time, and other fun activities that kept me grounded. I would especially like to thank my good friend Dr. Terrance Pendleton whose assistance and advice were invaluable when times got tough.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Preface

The need to store, compress and transmit digital data without the introduction of errors is as important today as it as ever been, if not more. If too many errors get into data then it may become corrupted, sometimes to the point where we can no longer gather useful information from it. While it is important to guard against errors, in some instances errors are inevitable. It may then be helpful to add redundant or parity data to the original data so that it can be recovered if errors are introduced. The process of adding redundant data so as to be able to recover the original data, if errors occur, is called error correcting codes. Error correcting codes is an essential tool in helping to mitigate the effects of noise and other issues that may introduce errors into data. They have been, and continue to be, used in a wide variety of applications from returning images from deep space, to ensuring clean crisp playback on optical disks. Big data analysis is a rapidly growing field of which the first step is to "clean up" the data. Error correcting decoding will then be an essential tool for any data scientist. Given that successful communication requires unambiguity, exact solutions are required whenever possible. Thus, symbolic computation and computer algebra systems provide the best framework for doing error correction.

In January 2015 the author joined a research project that was focused on Parametric Linear System Solving with Error Correction. At that time the method being employed was a generalization of the [Gemmell and Sudan 1992] description of the [Welch and Berlekamp 1986] decoder for [Reed and Solomon 1960] codes. The main results of the work done prior to the author joining the research project can be found in [Boyer and Kaltofen 2014] Symbolic Numeric Computation (SNC) paper. The author along with collaborators continued the work presented in [Boyer and Kaltofen 2014]. The main results of the continuation along with the answers to some of the questions that were raised as result of continuing the work of [Boyer and Kaltofen 2014] are the subjects of this document (the author's dissertation).

Chapter 2 gives an overview of the [Boyer and Kaltofen 2014] algorithm found in their

SNC paper. The algorithm computes the unique solution of a parametric linear system by first evaluating the system, then interpolating the parametric solution from the evaluation. See [McClellan 1973]. This algorithm is a generalization of the [Gemmell and Sudan 1992] description of the [Welch and Berlekamp 1986] decoder for [Reed and Solomon 1960] codes. The systems [Boyer and Kaltofen 2014] considered are full rank and overdetermined. The solution to such systems is a vector of rational functions. To enforce uniqueness, the solution is restricted to the one with the least common monic denominator. As mentioned in their SNC paper, there are two ways to view this algorithm. The first, and the main focus of the SNC paper, is numeric. That is the algorithm uses error correcting code techniques to account for ill-conditioned scalar matrices that result after the parametric systems have been evaluated. The second view is as a black box linear system solver that can correct true errors. True errors are scalar linear systems returned by the black box that do not agree with the evaluated solution. We will only consider the latter view in Chapter 2.

Chapter 2 continues with a description our early termination algorithms for parametric linear systems solving with error correction. This work is published in the proceedings of the 2017 International Symposium for Symbolic and Algebraic Computation (ISSAC). See [Kaltofen, Pernet, Storjohann, and Waddell 2017]. As mentioned earlier the [Boyer and Kaltofen 2014] algorithm is a generalization of the [Gemmell and Sudan 1992] description of the [Welch and Berlekamp 1986] decoder for [Reed and Solomon 1960] codes. Thus this algorithm requires the sum of a degree bound for the numerators plus a degree bound for the denominator of the solution. It is possible that the degree bounds that are input to the algorithm are much larger than the actual degrees. We thus describe an algorithm, that given the same inputs may compute the solution from fewer evaluations. Later in Chapter 2 we introduce a second count that can be used to recover the solution of black box parametric linear systems with errors. This new count is based on the work by [Cabay 1971]. See also [Olesh and Storjohann 2007]. The Cabay count does not rely on degree bounds for the solution, rather it uses degree bounds for the parametric system. The algorithms iterates until the Cabay Criterion is reached. At which point it returns the solution. We then compare the two counts and say exactly when the Cabay count is fewer than the generalized Welch/Berlekamp count. Next we combine our two early terminations strategies into one general early termination algorithm. We also use an error rate as well as a rank drop rate instead of fixed bounds for the number of errors and rank drops respectively. This allows for further reduction in the number of queries to the black box. A similar strategy can be found in [Kaltofen and Yang 2013] and [Kaltofen and Yang 2014].

The [Boyer and Kaltofen 2014] algorithm for parametric linear system solving with errors for full rank systems can be specialized to rational function vector recovery with errors. This idea is the subject of the Chapter 3. This work can also be found in our ISSAC 2017 paper. We show that if the rational function vector is the solution to a full rank linear system then our early terminations strategies apply. Thus we are sometimes able to recover the rational function vector using fewer evaluations than generalized Welch/Berlekamp decoding. If evaluations at poles (roots of the denominator) are allowed there are examples where the Cabay count is insufficient to recover the rational function vector. We show that if in addition to indicating that an evaluation is a pole the black box also gives information about the numerators of the solution then we are able to recover the solution.

In chapter 4 we describe an algorithm for parametric linear systems solving with error correction when the matrix of the system is rank deficient. Such systems have multiple solutions. We wish to compute a solution that agrees with the input degree bounds. In the full rank case the solution is unique and the solution of minimum degree has the error locator polynomial (a polynomial that has the error locations as its roots) as a factor. Unfortunately, this algorithm does not always return a solution that agrees with the degree bounds in the rank deficient case. The issue is that while the common monic denominator of minimum degree is unique there are many solutions with this denominator. The algorithm then does not always return a solution with the error locator polynomial as a factor. We know that if no errors occur the full rank algorithm works. The algorithm we describe for rank deficient systems first locates and removes the errors and we then able to compute a solution that agrees with the input degree bounds.

In the error model we used in Chapter 3 an errors means that one or more of the entries in the evaluated vector are wrong. While this model is reasonable, it is possible that errors affect the entire vector. We show in Chapter 5 that if errors affect the entire vector then we can solve an underdetermined system to find the solution and error locations simultaneously. We further show that if errors affect the entire vector then there is no need to assume that the erroneous values are random field elements.

In Chapter 6 we view the recovery of a vector of polynomials as the decoding of interleaved Reed-Solomon codes. It is known that interleaved schemes improve the error correction capabilities of codes when errors occur in bursts. See [Bleichenbacher, Kiayias, and Yung 2003] and [Schmidt, Sidorenko, and Bossert 2009]. We design a method for encoding a message as the coefficients of polynomials that is able to capitalize on the savings we describe in Chapter 5. We then test our technique on a Simplified Gilbert Channel, see [Yee and Weldon 1995], and present our initial findings

Chapter 7 contains as summary of the dissertation and Chapter 8 talks about possible future directions. The Appendix A that follow contain Maple programs that implement of our algorithms.

Chapter 2

Early Termination in Parametric Linear System Solving with Error Correction for Full Rank Systems

2.1 Introduction

Consistent linear systems of the form $A(u)x = b(u)$, where $A(u) \in \mathsf{K}[u]^{m\times n}$ and full rank, $b(u) \in \mathsf{K}[u]^m$, $m \geq n$, and K is a field, have as their solution rational functions $x_i = f^{[i]}(u)/g^{[i]}(u)$, $1 \leq i \leq n$. In particular there is a solution $\frac{1}{g(u)}f(u)$,[1] where $g(u)$ is the monic, least common denominator, that is

$$\mathrm{GCD}(f, g) \overset{\mathrm{def}}{=} \mathrm{GCD}(\mathrm{GCD}_i(f^{[i]}), g) = 1.$$

The solution of such a system can be determined by evaluating the system at distinct points $\xi_\ell \in \mathsf{K}$ and interpolating the evaluated solution [McClellan 1973]. The solution can be found even if some evaluations are erroneous. The matrices of the systems we consider have full column rank, so their solution in the form $\frac{1}{g(u)}f(u)$ is unique. Note that for full rank matrices with univariate polynomial entries there are finitely many $\xi_\ell \in \mathsf{K}$ that may cause the evaluated matrix to be rank deficient. If for each evaluation that causes the matrix with scalar entries to be rank deficient an extra evaluation is included, then techniques from algebraic error correcting codes can be used to compute the solution [Olshevsky and Shokrollahi 2003; Kaltofen and Pernet 2013; Kaltofen and Yang 2013, 2014; Boyer and Kaltofen 2014]. Furthermore in [Boyer and Kaltofen 2014] it is shown that for non-erroneous evaluation points, ξ_ℓ, it is not necessary to have $A(\xi_\ell)$ and $b(\xi_\ell)$ in order to interpolate the solution. Rather it is enough to have a scalar

[1]We write $\frac{1}{g}f$ if f is a vector of polynomials and $\frac{1}{g}$ a rational function scalar.

matrix $\hat{A}^{[\ell]}$ and right side vector $\hat{b}^{[\ell]}$ that have the evaluated solution $\frac{1}{g(\xi_\ell)} f(\xi_\ell)$ as a solution.

Consider the following model. Suppose there exists an oracle, which we will refer to as the black box. If we supply the black box with a value, ξ_ℓ, from the field K the black box returns to us $\hat{A}^{[\ell]}$ and $\hat{b}^{[\ell]}$ with entries from the field K. The scalar matrix, $\hat{A}^{[\ell]}$, and right side vector, $\hat{b}^{[\ell]}$, which are returned may not be $A(\xi_\ell)$ and $b(\xi_\ell)$. Nevertheless, if we query the black box L times we assume that $\leq E$ times we get $\hat{A}^{[\lambda]}$ and $\hat{b}^{[\lambda]}$ such that $\hat{A}^{[\lambda]} f(\xi_\lambda) \neq g(\xi_\lambda) \hat{b}^{[\lambda]}$. Such evaluations are considered to be erroneous. Furthermore we assume that fewer than R times the black box returns $\hat{A}^{[\ell]}$ and $\hat{b}^{[\ell]}$ such that $\hat{A}^{[\ell]} f(\xi_\ell) = g(\xi_\ell) \hat{b}^{[\ell]}$ but $\operatorname{rank}(\hat{A}^{[\ell]}) < n$. The objective is to find the solution $x = \frac{1}{g(u)} f(u)$ of the system $A(u)x = b(u)$ from as few queries of the black box as possible.

The count for the number of ξ_ℓ

$$L \geq L_{\text{BK}} \overset{\text{def}}{=} d_f + d_g + R + 2E + 1 \tag{2.1}$$

is employed by [Boyer and Kaltofen 2014] to recover the solution $\frac{1}{g(u)} f(u)$. The input parameters must satisfy the following specifications:

$$d_f \geq \deg(f) \overset{\text{def}}{=} \max_{1 \leq i \leq m} \{\deg(f^{[i]})\}, \quad d_g \geq \deg(g), \tag{2.2}$$

$$E \geq \big| \{ \lambda \mid \hat{A}^{[\lambda]} f(\xi_\lambda) \neq g(\xi_\lambda) \hat{b}^{[\lambda]} \text{ for } 0 \leq \lambda \leq L-1 \} \big|,^{2} \tag{2.3}$$

$$\begin{aligned} R \geq \big| \{ \ell \mid {} & \hat{A}^{[\ell]} f(\xi_\ell) = g(\xi_\ell) \hat{b}^{[\ell]} \\ & \text{and } \operatorname{rank}(\hat{A}^{[\ell]}) < n \text{ for } 0 \leq \ell \leq L-1 \} \big|. \end{aligned} \tag{2.4}$$

Here $|\cdot|$ denotes the cardinality of a set. The bounds E and R can be derived from an error and singularity rate; see below. If $n = m = 1$ and $\hat{A}^{[\ell]} = I_1$ and $\hat{b}^{[\ell]} = \frac{1}{g(\xi_\ell)} f(\xi_\ell)$ the algorithm is Welch/Berlekamp decoding of an algebraic (rational function) Reed-Solomon code [Welch and Berlekamp 1986]. We prove that for the vector rational function case if the input bounds in (2.2, 2.3, 2.4) are exact then the bound L_{BK} is tight; see Lemma 1.

If the bounds d_f and d_g on input significantly overestimate the degrees, by early termination we can reduce the number of required evaluations to

$$L^*_{\text{BK}} \overset{\text{def}}{=} \max\{d_f + \deg(g), d_g + \deg(f)\} + 2E^* + R^* + 1, \tag{2.5}$$

[2]Note that the condition (2.3) on the error bound E rules out inconsistent systems.

where

$$E^* \geq \big|\, \{\, \lambda \mid \hat{A}^{[\lambda]} f(\xi_\lambda) \neq g(\xi_\lambda) \hat{b}^{[\lambda]} \text{ for } 0 \leq \lambda \leq L^*_{\text{BK}} - 1 \,\} \,\big|, \tag{2.6}$$

$$R^* \geq \big|\, \{\, \ell \mid \hat{A}^{[\ell]} f(\xi_\ell) = g(\xi_\ell) \hat{b}^{[\ell]} \text{ and } \operatorname{rank}(\hat{A}^{[\ell]}) < n \text{ for } 0 \leq \ell \leq L^*_{\text{BK}} - 1 \,\} \,\big|. \tag{2.7}$$

The number of evaluations L^*_{BK} in (2.5) is determined iteratively, without $\deg(f)$ and $\deg(g)$ as input, but has to meet the conditions (2.6, 2.7) for the number of erroneous and rank-deficient systems at evaluation points ξ_ℓ. One can use the estimate $E^* = E$ and $R^* = R$ from (2.3,2.4) before, but we will show in Algorithm 2 below how to dynamically adjust E^* and R^* from an error and singularity rate associated with the black box for $\hat{A}^{[\ell]}, \hat{b}^{[\ell]}$, as is originally suggested in [Kaltofen and Yang 2014, Remark 1.1].

Following Stanely Cabay's [Cabay 1971] early termination strategy (see also [Olesh and Storjohann 2007]), we can derive a second count of number of evaluations. The new input parameters are specified as follows:

$$\left.\begin{aligned} d_A \geq \deg(A) &\overset{\text{def}}{=} \max\nolimits_{1\leq i\leq m, 1\leq j\leq n}\{\deg(a_{i,j})\},\\ d_b \geq \deg(b) &\overset{\text{def}}{=} \max\nolimits_{1\leq i\leq m}\{\deg(b_i)\}.\end{aligned}\right\} \tag{2.8}$$

Because in our algorithms we do not reconstruct A and b, for the bounds d_A and d_b we can use that pair $(A(u), b(u))$ with $A(u)f(u) = g(u)b(u)$ with a minimum $\deg(A)$. We derive a second evaluations count,

$$L^*_{\text{CAB}} = \max\{d_A + \deg(f), d_b + \deg(g)\} + 2E^* + R^* + 1, \tag{2.9}$$

for recovering the solution. Here E^* and R^* bound from above the corresponding counts for erroneous and singular systems in (2.3, 2.4) with L^*_{CAB} replacing L^*_{BK}. We prove that if all input parameter bounds are exact and $\deg(g) > \deg(A)$ then $L^*_{\text{CAB}} < L^*_{\text{BK}}$.

Next we combine the L^*_{BK} count and the L^*_{CAB} count into a general early termination strategy. This algorithm computes the solution using as few evaluations as possible when it is unclear how the $\deg(g)$ compares to the $\deg(A)$.

2.2 Exact Vector of Function Solving

We describe and prove an early termination algorithm for the exact vector of function solving algorithm in [Boyer and Kaltofen 2014]. Their algorithm solves a system of linear equations

$$A(u)\,x = b(u) \tag{2.10}$$

where $A(u) \in \mathsf{K}[u]^{m\times n}, b(u) \in \mathsf{K}[u]^m, m \geq n$ and K is a field. The system is assumed to have a unique solution

$$x = \begin{bmatrix} \vdots \\ \frac{1}{g(u)} f^{[i]}(u) \\ \vdots \end{bmatrix} \in \mathsf{K}(u)^n, \quad g \neq 0, \tag{2.11}$$

where g is the monic least common denominator. If for all $i, f^{[i]} = 0$ then g is set to 1. The solution vector x is computed by:

1. Selecting $L = d_f + d_g + R + 1$ distinct elements $\xi_\ell \in \mathsf{K}$ where
   (a) $0 \leq \ell \leq L-1$ and $\xi_{\ell_1} \neq \xi_{\ell_2}$ for $\ell_1 \neq \ell_2$.
   (b) $d_f \geq \deg(f)$.
   (c) $d_g \geq \deg(g)$.
   (d) $R \geq \big|\{\ell \mid \operatorname{rank}(A(\xi_\ell)) < n = \operatorname{rank}(A(u))\}\big|$.
2. Solving the homogeneous linear system

$$A(\xi_\ell) \begin{bmatrix} \vdots \\ \Phi^{[i]}(\xi_\ell) \\ \vdots \end{bmatrix} - \Psi(\xi_\ell) b(\xi_\ell) = 0, \tag{2.12}$$

where for all $i, \deg(\Phi^{[i]}) \leq d_f$ and $\deg(\Psi) \leq d_g$. The system (2.12) is linear in the coefficients of $\Phi^{[i]}(u)$ and $\Psi(u)$. There are $n(d_f+1) + d_g + 1$ unknown coefficients for $\Phi^{[i]}$ and Ψ and mL equations.

Theorem 1 [Boyer and Kaltofen 2014] *We suppose that for* $\geq d_f + d_g + 1$ *of the* ξ_ℓ *we have* $\operatorname{rank}(A(\xi_\ell)) = \operatorname{rank}(A(u)) = n$. *Let* $\Psi_{\min}$ *be the denominator component of a solution of* (2.12) *with* $\Psi_{\min} \neq 0$ *and scaled to have leading coefficient* 1 *in* u, *and of minimal degree of all such solutions, and let* $\Phi^{[i]}_{\min}$ *be the corresponding numerator components of that solution. Then for all* i *we have* $\Phi^{[i]}_{\min} = f^{[i]}$ *and* $\Psi_{\min} = g$.

The linear system (2.12) uses evaluations of $A(u)$ and $b(u)$ to solve for $x = \frac{1}{g}f$. The authors in [Boyer and Kaltofen 2014] show that it is not necessary to have the evaluations of $A(u)$ and $b(u)$ in order to solve (2.10). Rather it is enough, for each ξ_ℓ, to have a scalar matrix $\hat{A}^{[\ell]} \in \mathsf{K}^{m\times n}$ and right side vector $\hat{b}^{[\ell]} \in \mathsf{K}^m$ such that $\hat{A}^{[\ell]}f(\xi_\ell) = g(\xi_\ell)\hat{b}^{[\ell]}$. They also show that the solution can be computed even if some of the scalar matrices $\hat{A}^{[\ell]}$ and/or right side vectors $\hat{b}^{[\ell]}$ are erroneous. That is for some $0 \le \lambda \le L-1$,

$$\hat{A}^{[\lambda]}f(\xi_\lambda) \neq g(\xi_\lambda)\hat{b}^{[\lambda]}. \tag{2.13}$$

The solution is computed by:

1. Selecting $L \ge L_{\text{BK}} = d_f + d_g + R + 2E + 1$ distinct elements $\xi_\ell \in \mathsf{K}$ where
   (a) $R \ge \big|\{\ell \mid \operatorname{rank}(A(\xi_\ell)) < n \text{ and } \hat{A}^{[\ell]}f(\xi_\ell) = g(\xi_\ell)\hat{b}^{[\ell]}, 0 \le \ell \le L-1\}\big|$, that is (2.4).
   (b) $E \ge \big|\{\lambda \mid \hat{A}^{[\lambda]}f(\xi_\lambda) \neq g(\xi_\lambda)\hat{b}^{[\lambda]}, 0 \le \lambda \le L-1\}\big|$, that is (2.3).
2. Solving the homogeneous linear system

$$\hat{A}^{[\ell]}\begin{bmatrix} \vdots \\ \Phi^{[i]}(\xi_\ell) \\ \vdots \end{bmatrix} - \Psi(\xi_\ell)\hat{b}^{[\ell]} = 0,\ 1 \le i \le m, 0 \le \ell \le L-1, \tag{2.14}$$

where for all $i, \deg(\Phi^{[i]}) \le d_f + E$ and $\deg(\Psi) \le d_g + E$. The system (4.12) is linear in the coefficients of $\Phi^{[i]}(u)$ and $\Psi(u)$. There are $n(d_f + E + 1) + d_g + E + 1$ unknown coefficients of $\Phi^{[i]}(u)$ and $\Psi(u)$ and mL_{BK} equations.

Theorem 2 [Boyer and Kaltofen 2014] *We suppose that for $\le E$ of the ξ_ℓ we have $\hat{A}^{[\ell]}f(\xi_\ell) \neq g(\xi_\ell)\hat{b}^{[\ell]}$ and for $\ge d_f + d_g + E + 1$ of the ξ_ℓ we have $\operatorname{rank}(\hat{A}^{[\ell]}) = n$ and $\hat{A}^{[\ell]}f(\xi_\ell) = g(\xi_\ell)\hat{b}^{[\ell]}$. Let $\Psi_{\min}$ be the denominator component of a solution of (4.12) with $\Psi_{\min} \neq 0$ and scaled to have leading coefficient 1 in u, and of minimal degree of all such solutions, and let $\Phi^{[i]}_{\min}$ be the corresponding numerator components of that solution. Furthermore, let $\Lambda(u) = \prod_{\mu \text{ subj. to (2.13)}}(u - \xi_{\lambda_\mu})$ be an error locator polynomial. Then for all i we have $\Phi^{[i]}_{\min} = \Lambda f^{[i]}$ and $\Psi_{\min} = \Lambda g$.*

Remark 1 We assume we have a black box that we can probe with ξ_ℓ's. For each ξ_ℓ the black box returns $\hat{A}^{[\ell]}$ and $\hat{b}^{[\ell]}$. The scalar matrix $\hat{A}^{[\ell]}$ and scalar right-side vector $\hat{b}^{[\ell]}$ may not be $A(\xi_\ell)$ nor $b(\xi_\ell)$ respectively, but we are guaranteed that fewer than E are subject to condition (2.13). By Theorem 2, we can find the solution $x = \frac{1}{g}f$ as well as an error locator polynomial $\Lambda(u)$ that has as its roots the ξ_λ's that satisfy inequation (2.13). $\square$

2.3 Early Termination

In the black box model it is not possible to determine degree bounds for the solution a-priori. Thus it is possible that the degree bounds d_f and d_g are much larger than $\max_{1\le i\le n}\deg(f^{[i]})$ and $\deg(g)$ respectively. We describe next an algorithm that either finds the solution or determines that we need more evaluations. This allows us to design Algorithm 2, that computes the solution with possibly fewer evaluations than is required by the L_{BK} bound.

Algorithm 1: Compute $\frac{1}{g}f$ and Λ or determine degree bounds are too small.

Input: $d_f \ge \deg(f)$, $d_g \ge \deg(g)$, $0 \le d_f^* \le d_f$, $0 \le d_g^* \le d_g$,
$R^* \ge \big|\{\,\ell \mid \hat{A}^{[\ell]}f(\xi_\ell) = g(\xi_\ell)\hat{b}^{[\ell]}$ and $\operatorname{rank}(\hat{A}^{[\ell]}) < n$ for $0 \le \ell \le L_{\text{BK}}^* - 1\,\}\big|$,
$E^* \ge \big|\{\,\lambda \mid \hat{A}^{[\lambda]}f(\xi_\lambda) \ne g(\xi_\lambda)\hat{b}^{[\lambda]}$ for $0 \le \lambda \le L_{\text{BK}}^*-1\,\}\big|$,
with L_{BK}^* from Step 1 below,
a stream $(\hat{A}^{[\ell]}, \hat{b}^{[\ell]})$, $\ell = 0, 1, \ldots$ which is static on multiple calls and extensible in length on demand.

Output: $\frac{1}{g}f$ and Λ or "$\deg(f) > d_f^*$ and/or $\deg(g) > d_g^*$."

```
1: $L_{\text{BK}}^* \leftarrow \max\{d_f + d_g^*, d_g + d_f^*\} + R^* + 2E^* + 1$
2: Determine the null space of
```

$$\hat{A}^{[\ell]}\Phi^*(\xi_\ell) - \Psi^*(\xi_\ell)\hat{b}^{[\ell]} = 0, \quad \ell = 0, 1, \ldots, L_{\text{BK}}^* - 1, \tag{2.15}$$

```
   where $\deg(\Phi^*) \le d_f^* + E^*, \deg(\Psi^*) \le d_g^* + E^*$
3: **if** only trivial solution **then**
       **return** "$\deg(f) > d_f^*$ and/or $\deg(g) > d_g^*$"; **end if**
4: Compute a basis, $B$, for the null space.
5: Compute the column echelon form for $B, \mathrm{CEF}(B)$.
   Retrieve the last column,
```

$$\mathrm{CEF}(B)_{*,r} \leftarrow \begin{bmatrix} \overrightarrow{\Psi^*_{\min}} \\ \overrightarrow{\Phi^{*\,[1]}_{\min}} \\ \vdots \\ \overrightarrow{\Phi^{*\,[m]}_{\min}} \end{bmatrix}, \text{ which has } \Psi^*_{\min} \ne 0.$$

```
   Here $\vec{\cdot}$ are coefficient vectors.
6: $\Lambda^* \leftarrow \mathrm{GCD}(\Phi^*_{\min}, \Psi^*_{\min})$; $k^* \leftarrow \deg(\Lambda^*)$.
7: $(f^*, g^*) \leftarrow (\frac{1}{\Lambda^*}\,\Phi^*_{\min}, \Psi^*_{\min}/\Lambda^*)$.
8: **if** $\deg(f^*) > d_f^*$ or $\deg(g^*) > d_g^*$ or $k^* > E^*$ **then**
```

```
    return "deg($f$) > $d_f^*$ and/or deg($g$) > $d_g^*$"; end if
9: return $f \leftarrow f^*$, $g \leftarrow g^*$, $\Lambda \leftarrow \Lambda^*$; end if
```

Observe that Algorithm 1 is similar to the algorithm implied by Theorem 2. The main difference is that it uses the $L_{\text{BK}}^* \leq L_{\text{BK}}$ count. Recall that Theorem 2 requires $\geq L_{\text{BK}}$ evaluations to find the solution. We use the results of Theorem 2 to prove the correctness of our algorithm. That is our algorithm either determines that we just computed an interpolant of the evaluation points or that we have indeed found the solution. Recall that we assume there exists a unique solution to equation (2.10).

In Step 2 we compute a solution similar to (4.12). The difference being that we use the starred bounds. Observe that if $\deg(f) \leq d_f^*$ and $\deg(g) \leq d_g^*$ and we were to substitute $d_f = d_f^*$, $d_g = d_g^*$ in L_{BK}, then with $L_{\text{BK}}^* \geq d_f^* + d_g^* + 2E^* + R^*$ by Theorem 2 we are guaranteed to find the solution $(\Lambda f, \Lambda g)$. So if B indicates there is only the trivial solution then it must be the case that $\deg(f) > d_f^*$ and/or $\deg(g) > d_g^*$.

In Step 5 we compute a non-zero polynomial Ψ^* of minimal degree ($\Psi_{\text{min}}^* \neq 0$). We claim that the last column of CEF(B) contains Ψ_{min}^*. The fact that the degree of Ψ_{min}^* is minimum is clear from the form of the CEF(B). To see why $\Psi_{\text{min}}^* \neq 0$, assume that $\Psi_{\text{min}}^* = 0$. Then for all ξ_ℓ, $\hat{A}^{[\ell]}\Phi_{\text{min}}^*(\xi_\ell) = \Psi_{\text{min}}^*(\xi_\ell)\hat{b}^{[\ell]} = 0^m$. On $\geq \max\{d_f + d_g^*, d_g + d_f^*\} + E^* + 1$ evaluations $\text{rank}(\hat{A}^{[\ell]}) = n$, that is $\Phi_{\text{min}}^*(\xi_\ell) = 0$, which implies by $\deg(\Phi_{\text{min}}^*) \leq d_f^* + E^*$ that $\Phi_{\text{min}}^* = 0$. This cannot be since CEF(B) is a basis for the solution space of equation (2.15) and thus cannot contain the zero vector. Hence $\Psi_{\text{min}}^* \neq 0$.

In Step 7 we define $\frac{1}{g^*}f^* = \frac{1}{\Psi_{\text{min}}^*}\Phi_{\text{min}}^*$. We think of $\frac{1}{g^*}f^*$ as our candidate solution. Next in Step 8 we check if the candidate solution agrees with our starred bounds. We know from Theorem 2 that if $d_f^* \geq \deg(f)$ and $d_g^* \geq \deg(g)$ the bounds for the minimal solutions must be satisfied, so if they fail at least one bound is wrong.

Finally, we claim that if Algorithm 1 returns at Step 9 then we have computed the solution $\frac{1}{g}f$. Of the L_{BK}^* points ξ_ℓ at Step 9 we discard $\leq R^*$ "good" rank drops and $\leq E^*$ erroneous points for the solution (f, g) and $\leq k^* = \deg(\Lambda^*) \leq E^*$ points ξ_ℓ that have $\Lambda^*(\xi_\ell) = 0$. The remaining $\geq \max\{d_f + d_g^*, d_g + d_f^*\} + 1$ distinct ξ_ℓ satisfy

1. $\text{rank}(\hat{A}^{[\ell]}) = n$,
2. $\hat{A}^{[\ell]}f(\xi_\ell) = g(\xi_\ell)\hat{b}^{[\ell]}$,
3. $\hat{A}^{[\ell]}f^*(\xi_\ell) = g^*(\xi_\ell)\hat{b}^{[\ell]}$, because

$$\begin{aligned}\hat{A}^{[\ell]}\Phi_{\text{min}}^*(\xi_\ell) &= \hat{A}^{[\ell]}\Lambda^*(\xi_\ell)f^*(\xi_\ell)\\ &= \Psi_{\text{min}}^*(\xi_\ell)\hat{b}^{[\ell]} = \Lambda^*(\xi_\ell)g^*(\xi_\ell)\hat{b}^{[\ell]},\end{aligned}$$

and $\Lambda^*(\xi_\ell) \neq 0$.

From Items 2 and 3 we get $\hat{A}^{[\ell]}(g(\xi_\ell)f^*(\xi_\ell) - g^*(\xi_\ell)f(\xi_\ell)) = 0$ which by Item 1 yields $g(\xi_\ell)f^*(\xi_\ell) - g^*(\xi_\ell)f(\xi_\ell) = 0$, that for at least $\max\{d_f + d_g^*, d_g + d_f^*\} + 1$ distinct ξ_ℓ. The vector $(gf^* - g^*f)(u)$ has polynomials of degree $\leq \max\{d_f + d_g^*, d_f^* + d_g\}$ and is therefore equal 0, which proves $\frac{1}{g^*}f^* = \frac{1}{g}f$.

We observe that $d_f^* \leq d_f$ and $d_g^* \leq d_g$ implies that $L_{\text{BK}}^* \leq L_{\text{BK}}$. Now Algorithm 1 guarantees that with L_{BK}^* many evaluations we either compute the solution $\frac{1}{g}f$ or we determine that $\deg(f) > d_f^*$ and/or $\deg(g) > d_g^*$. Thus L_{BK}^* count can be used in an early termination strategy. We give the details in the following algorithm.

Algorithm 2: Early Termination Strategy.

```
Input: $d_f \geq \deg(f)$, $d_g \geq \deg(g)$,
    $\rho_E < 1/2$, a rational number with denominator $q_E$,
            # the error rate.
    $\rho_R < 1 - 2\rho_E$, a rational number with denominator $q_R$,
            # the rank drop rate, see Remark 2.
    # $q_E = q_R = \infty$ is permissible but may require
    # more evaluations.
Output: $\frac{1}{g}f$ and $\Lambda$.
1: $d_f^* \leftarrow 0$; $d_g^* \leftarrow 0$.
2: $D \leftarrow \max\{d_f + d_g^*, d_g + d_f^*\} + 1$.
3: $E^* \leftarrow \lfloor \bar{E}^* \rfloor$; $R^* \leftarrow \lfloor \bar{R}^* \rfloor$ with
```

$$\bar{E}^* = \frac{1}{1-2\rho_E-\rho_R}\Big(\rho_E\big(D+1-\frac{1}{q_R}\big)+(1-\rho_R)\big(1-\frac{1}{q_E}\big)\Big). \tag{2.16}$$

$$\bar{R}^* = \frac{1}{1-2\rho_E-\rho_R}\Big(\rho_R\big(D+2-\frac{2}{q_E}\big)+(1-2\rho_E)\big(1-\frac{1}{q_R}\big)\Big). \tag{2.17}$$

```
4: if Algorithm 1($d_f, d_g, d_f^*, d_g^*, E^*, R^*$) returns at Step 9
       then return $\frac{1}{g}f$; end if
5: while(true) $D \leftarrow D + 1$.
   # returns below for $D = \max\{d_f + \deg(g), d_g + \deg(f)\} + 1$
6:     Reassign $E^*, R^*$ as in Step 3 using the updated $D$ in (2.16, 2.17).
7:     for all $(d_f^*, d_g^*)$ with $D = \max\{d_f + d_g^*, d_g + d_f^*\} + 1$ do
8:         if Algorithm 1($d_f, d_g, d_f^*, d_g^*, E^*, R^*$)
               returns at Step 9 then return $(f, g, \Lambda)$; end if
   end for end while
```

Remark 2 Algorithm 2 saves evaluations is two ways. The first way we save evaluations is by probabilistic computation of E^* and R^* based on the size of D rather than using fixed bounds. Like [Kaltofen and Yang 2013] we view evaluations as probing a black box, thus we can also relate the error rate of the black box to E^*. Also given the number of evaluation and a strategy for choosing the evaluation points one may have a rate at which the problem drops rank. Such a rate for the rank drop can then be related to R^*. If there is no such rate then R from the L_{BK} count can always be substituted for R^* without affecting Algorithm 2.

We make the following assumption on the input error rates:

Assumption 1 *Suppose that for $L \geq L_E^{\min}$ the number of erroneous evaluations, k_E,* ***always*** *satisfies $k_E \leq \lceil \rho_E L \rceil$, and also for $L \geq L_R^{\min}$: $k_R \leq \lceil \rho_R L \rceil$ evaluations give rise to valid but rank deficient systems.*

Here $L_E^{\min}$ and $L_R^{\min}$ are sufficiently large numbers of evaluations for which the assumptions on k_E and k_R are sensible. Let $L_{\min} = \max\{L_E^{\min}, L_R^{\min}\}$, then $L_{\min}$ is a minimum on the number of evaluations our algorithm can work with. Assumption 1 differs from the rate assumptions in [Kaltofen and Yang 2013, Remark 1.1] and [Kaltofen and Yang 2014, Remark 1.1, Lemma 3.1] in that there we suppose $k_E \leq \lfloor \rho_E L \rfloor$, which implies no error for $L < 1/\rho_E$. Our assumption here allows 1 error. Note that for $\rho_R = 0$, $q_R = \infty$ and $\rho_E = 1/q_E$ we get $\bar{E}^* = D/(q_E - 2) + q_E/(q_E - 2)$ whereas in [Kaltofen and Yang 2013, 2014] we have $\bar{E}^* = D/(q_E - 2)$. In [Kaltofen and Yang 2014, Remark 1.1] the assumptions are probabilistically validated by adjusting the error rate upwards and bounding the probability of failure via Chernoff bounds.

We now show that Assumption 1 and the computation of E^* and R^* in (2.16, 2.17) guarantee the input specifications for Algorithm 1. We have

$$\begin{aligned}\bar{L}^* &= D + 2\big(\rho_E \bar{L}^* + 1 - \frac{1}{q_E}\big) + \rho_R \bar{L}^* + 1 - \frac{1}{q_R}\\ &= \frac{1}{1-2\rho_E-\rho_R}\big(D + 3 - \frac{2}{q_E} - \frac{1}{q_R}\big)\end{aligned}$$

and for $\bar{E}^*, \bar{R}^*$ in (2.16,2.17) we have

$$\bar{E}^* = \rho_E \bar{L}^* + 1 - \frac{1}{q_E},\ \bar{R}^* = \rho_R \bar{L}^* + 1 - \frac{1}{q_R},\ \bar{L}^* = D + 2\bar{E}^* + \bar{R}^*.$$

Therefore we have

$$\begin{aligned}
k_E^* &\leq \lceil \rho_E L_{\text{BK}}^* \rceil = \lceil \rho_E (D + 2E^* + R^*) \rceil \\
&\leq \rho_E (D + 2E^* + R^*) + 1 - \frac{1}{q_E} \\
&\leq \rho_E (D + 2\bar{E}^* + \bar{R}^*) + 1 - \frac{1}{q_E} \\
&= \rho_E \bar{L}^* + 1 - \frac{1}{q_E} = \bar{E}^*,
\end{aligned}$$

which implies by the integrality of k_E^* that $k_E^* \leq \lfloor \bar{E}^* \rfloor = E^*$, as is required by Algorithm 1. Similarly, one proves $k_R^* \leq R^*$.

We discuss now the second way Algorithm 2 saves evaluations. The algorithm initializes d_f^* and d_g^* to zero. Thus $L_{\text{BK}}^* \leq L_{\text{BK}}$. The fewest number of evaluations we can use in Algorithm 1 is $D + R^* + E^*$ where $D = \max\{d_f, d_g\} + 1$. Note this is the first bound used by Algorithm 2. We assume that $D \geq L$, we can always adjust d_f and/or d_g so that $D \geq L$. If L_{BK}^* has too few evaluations to return the solution, D is incremented by 1 and R^* and E^* are adjusted if needed. The algorithm then tries all possible combinations of d_f^* and d_g^* such that $D = \max\{d_f + d_g^*, d_g + d_f^*\} + 1$. Thus we find the solution while incrementing D as slowly as possible. □

2.4 Cabay Early Termination

We now describe the count L^*_{CAB} that incorporates degree bounds for the system being solved. The count is based on work in [Cabay 1971] (see also [Olesh and Storjohann 2007]). In Theorem 3, given exact values for degree parameters, we give the criteria and proof for when $L^*_{\text{CAB}} < L^*_{\text{BK}}$.

Consider another count L^*_{CAB},

$$L^*_{\text{CAB}} = \max\{d_A + d^*_f, d_b + d^*_g\} + R^* + 2E^* + 1,$$

where $d_A \geq \deg(A)$ and $d_b \geq \deg(b)$. See (2.8) for the definitions of $\deg(A)$ and $\deg(b)$. Similar to Algorithm 1 we present next an algorithm that uses the L^*_{CAB} bound and either determines one of the starred bounds is too small or returns the solution.

Algorithm 3: Cabay Early Termination

```
Input: $d_A \geq \deg(A)$, $d_b \geq \deg(b)$,
    $d^*_f$, $d^*_g$, with $0 \leq d^*_f \leq \deg(f)$, $0 \leq d^*_g \leq \deg(g)$
        # same as in Algorithm 1
    $R^* \geq \big| \{ \ell \mid \hat{A}^{[\ell]} f(\xi_\ell) = g(\xi_\ell)\hat{b}^{[\ell]}$
                and $\text{rank}(\hat{A}^{[\ell]}) < n \text{ for } 0 \leq \ell \leq L^*_{\text{CAB}} - 1 \} \big|$,
    $E^* \geq \big| \{ \lambda \mid \hat{A}^{[\lambda]} f(\xi_\lambda) \neq g(\xi_\lambda)\hat{b}^{[\lambda]} \text{ for } 0 \leq \lambda \leq L^*_{\text{CAB}} - 1 \} \big|$,
Output: $\frac{1}{g} f$ and $\Lambda$ or "$\deg(f) > d^*_f$ and/or $\deg(g) > d^*_g$".
1: $L^*_{\text{CAB}} \leftarrow \max\{d_A + d^*_f, d_b + d^*_g\} + R^* + 2E^* + 1$.
2: Determine the null space of the system

    $$\hat{A}^{[\ell]}\Phi^*(\xi_\ell) - \Psi^*(\xi_\ell)\hat{b}^{[\ell]} = 0,\ \ell = 0, 1, \ldots, L^*_{\text{CAB}} - 1, \quad (2.18)$$

   where $\deg(\Phi^*) \leq d^*_f + E^*, \deg(\Psi^*) \leq d^*_g + E^*$.
3: if only the trivial solution then
4:     return $\deg(f) > d^*_f$ and/or $\deg(g) > d^*_g$; end if
5: Compute a basis, $B$, for the null space
6: Compute the column echelon form for $B, \text{CEF}(B)$. See Step 7 in Algorithm 1.
7: $\Lambda^* \leftarrow \text{GCD}(\Phi^*_{\min}, \Psi^*_{\min})$; $k^* \leftarrow \deg(\Lambda^*)$.
8: $(f^*, g^*) \leftarrow (\frac{1}{\Lambda^*}\Phi^*_{\min}, \Psi^*_{\min}/\Lambda^*)$.
9: return $f \leftarrow f^*$, $g \leftarrow g^*$, and $\Lambda \leftarrow \Lambda^*$.
```

In Step 2 we again compute a similar object to (4.12) using our new starred degree bounds . We now justify Steps 3 and 4. We prove that if the computation in Step 2 produces only the trivial solution then $\deg(f) > d_f^*$ and/or $\deg(g) > d_g^*$. Assume $\deg(f) \leq d_f^*$ and $\deg(g) \leq d_g^*$. Then $(\Phi^*, \Psi^*) = (\Lambda f, \Lambda g)$ solves (2.18). Thus equation (2.18) cannot only contain the trivial solution. This implies that if (2.18) has only the trivial solution then $\deg(f) > d_f^*$ and/or $\deg(g) > d_g^*$.

We now justify Step 9. We prove that $\frac{1}{g^*} f^*$ is the solution of our system. Furthermore, the $\mathrm{GCD}(\Phi^*_{\min}, \Psi^*_{\min})$ is the error locator polynomial. If we are at Step 9 of our algorithm then we have that on at least $\max\{d_A + d_f^*, d_g^* + d_b\} + E^* + 1$ evaluations $\hat{A}^{[\ell]} f(\xi_\ell) = g(\xi_\ell)\hat{b}^{[\ell]}$ and $\mathrm{rank}(\hat{A}^{[\ell]}) = n$. The latter implies that $g(\xi_\ell) \neq 0$, for otherwise $f(\xi_\ell) = 0$ and $\frac{1}{g} f$ would not be reduced. For those ℓ we have computed Φ^* and Ψ^* such that $\hat{A}^{[\ell]}\Phi^*(\xi_\ell) = \Psi^*(\xi_\ell)\hat{b}^{[\ell]}$.

We show first that $\hat{A}^{[\ell]}\Phi^*(\xi_\ell) = \Psi^*(\xi_\ell)\hat{b}^{[\ell]}$ implies $A(\xi_\ell)\Phi^*(\xi_\ell) = \Psi^*(\xi_\ell)b(\xi_\ell)$. If $\Psi(\xi_\ell) = 0$ then $\Phi(\xi_\ell) = 0$ because $\hat{A}^{[\ell]}$ has linearly independent columns. If on the other hand $\Psi(\xi_\ell) \neq 0$ we get $\Phi^*(\xi_\ell)/\Psi^*(\xi_\ell) = f(\xi_\ell)/g(\xi_\ell)$ since the solution is unique. Now $A(\xi_\ell)(f(\xi_\ell)/g(\xi_\ell)) = A(\xi_\ell)(\Phi^*(\xi_\ell)/\Psi^*(\xi_\ell)) = b(\xi_\ell)$. So the computed Φ^* and Ψ^* must satisfy $A(\xi_\ell)\Phi^*(\xi_\ell) = \Psi^*(\xi_\ell)b(\xi_\ell)$.

Since $A(u)\Phi^*(u) - \Psi^*(u)b(u)$ is a polynomial vector of degree $\leq \max\{d_A + d_f^*, d_b + d_g^*\} + E^*$ it is uniquely determined by $\max\{d_A + d_f^*, d_b + d_g^*\} + E^* + 1$ distinct evaluation points so we have $A(u)\Phi^*(u) = \Psi^*(u)b(u)$. So $\frac{1}{g} f = \frac{1}{\Psi^*_{\min}}\Phi^*_{\min} = \frac{1}{g^*} f^*$. This implies there is a polynomial $\Lambda^*(u)$ with $\Lambda^* f = \Phi^*_{\min}$ and $\Lambda^* g = \Psi^*_{\min}$. For each λ we have $\hat{A}^{[\lambda]} f(\xi_\lambda) \neq g(\xi_\lambda)\hat{b}^{[\lambda]}$ and $\hat{A}^{[\lambda]}(\Lambda^* f)(\xi_\lambda) = \hat{A}^{[\lambda]}\Phi^*_{\min}(\xi_\lambda) = \Psi^*_{\min}(\xi_\lambda)\hat{b}^{[\lambda]} = (\Lambda^* g)(\xi_\lambda)\hat{b}^{[\lambda]}$ which implies $\Lambda^*(\xi_\lambda) = 0$. Thus $\Lambda = \Lambda^*$.

Remark 3 Any non-zero solution computed in Step 5 of the previous algorithm has the property $\frac{1}{g} f = \frac{1}{\Psi^*}\Phi^*$. Nevertheless, only the pair $(\Phi^*_{\min}, \Psi^*_{\min}) = (\Lambda f, \Lambda g)$. So if there is no need to compute the error locator polynomial then Step 6 is unnecessary.

Remark 4 If we implement Algorithm 2 replacing Algorithm 1 with Algorithm 3 then we then get an early termination strategy for Cabay Termination. □

Remark 5 The matrix $A(u)$ having full rank implies by Cramer's rule that we can set $d_f = (n-1)d_A + d_b$ and $d_g = nd_A$. So $L_{\text{CAB}} \geq nd_A + d_b + R + 2E + 1 = d_f + d_g/n + R + 2E + 1$ in comparison to Theorem 2, which has $L_{\text{BK}} \geq d_f + d_g + R + 2E + 1$. In Theorem 3 we generalize when L_{CAB} is better than L_{BK}. □

Theorem 3 *If all bounds are exact then $L_{\text{CAB}} < L_{\text{BK}}$ if and only if $\deg(g) > \deg(A)$.*

Proof. $Af = gb$ implies $\deg(Af) = \deg(gb) = \deg(g) + \deg(b)$. Since some terms can cancel due to the matrix vector multiplication, Af, we have $\deg(Af) \leq \deg(A) + \deg(f)$. This implies that $\deg(g) + \deg(b) \leq \deg(A) + \deg(f)$.

Assume $\deg(g) + \deg(b) < \deg(A) + \deg(f)$. Then $L_{\text{CAB}} = \deg(f) + \deg(A) + R + 2E + 1 < L_{\text{BK}} = \deg(f) + \deg(g) + R + 2E + 1$if and only if $\deg(g) > \deg(A)$.
Now assume $\deg(g) + \deg(b) = \deg(A) + \deg(f)$, then there are two cases.
Case 1: $L_{\text{CAB}} = \deg(f) + \deg(A) + R + 2E + 1$.
Case 2: $L_{\text{CAB}} = \deg(g) + \deg(b) + R + 2E + 1$.
We have already dealt with case 1. Consider case 2, $L_{\text{CAB}} = \deg(g) + \deg(b) + R + 2E + 1 < L_{\text{BK}} = \deg(f)+\deg(g)+R+2E+1$ if and only if $\deg(b) < \deg(f)$. This implies $\deg(g) > \deg(A)$ since we assumed that $\deg(g) + \deg(b) = \deg(A) + \deg(f)$. $\square$

Remark 6 If $n = m = 1$ then the Cramer rule bound in Remark 5 yields, in the exact case, $L_{\text{CAB}} = L_{\text{BK}}$. In fact the linear system $A(u)x = b(u)$ is actually of the form $a(u)x = b(u)$ where $a(u), b(u) \in K[u]$. This implies $x = b(u)/a(u) = f/g$ which implies $a(u) = h(u)g(u)$ and $b(u) = h(u)f(u)$, where $h(u) \in K[u]$. Thus if we use the exact degrees for our bounds we get $L_{\text{BK}} \le L_{\text{CAB}}$, since in this case $\deg(g) \le \deg(A)$. Furthermore, if one uses fewer than $L = \deg(f) + \deg(g) + 2k + 1$ evaluations then one loses the guarantee of a unique solution. In Lemma 1 below, given only $L = \deg(f) + \deg(g) + 2k$ we construct a second solution. $\square$

Lemma 1 *Let $n = m = 1$ and K a field. For all $f, g \in \mathsf{K}[u]$ with $\deg(g) \ge 1$ and $\mathrm{GCD}(f, g) = 1$ and for all $\xi_0, \dots, \xi_{L-1}$ with $L = \deg(f)+\deg(g)+2k, \xi_\ell \ne 0, \xi_{\ell_1} \ne \xi_{\ell_2}$ for $\ell_1 \ne \ell_2, 0 \le \ell, \ell_1, \ell_2 \le L - 1$ and $g(\xi_\ell) \ne 0$ for all ℓ with $0 \le \ell \le \deg(f) + \deg(g) - 1$ and for all $k \ge 0$ we have: if $|\mathsf{K}| \ge 2(\deg(f) + \deg(g) + k) + 1$ then there exist $\bar f, \bar g \in \mathsf{K}[u]$ and there exist $\hat a^{[\ell]}, \hat b^{[\ell]} \in \mathsf{K}$ for all ℓ with $0 \le \ell \le L - 1$ such that*

1. *$f/g \ne \bar f/\bar g$, $\mathrm{GCD}(\bar f, \bar g) = 1$, $\deg(f) = \deg(\bar f)$ and $\deg(g) = \deg(\bar g)$.*
2. *$\bar g(\xi_\ell) \ne 0$ for all ℓ with $0 \le \ell \le \deg(f) + \deg(g) - 1$.*
3. *$\hat a^{[\ell]} f(\xi_\ell) = g(\xi_\ell)\hat b^{[\ell]}$ for all ℓ with $0 \le \ell \le \deg(f) + \deg(g) + k - 1$,*
   $\hat a^{[\ell]} \bar f(\xi_\ell) = \bar g(\xi_\ell)\hat b^{[\ell]}$ for all ℓ with $0 \le \ell \le \deg(f) + \deg(g) - 1$ or $\deg(f) + \deg(g) + k \le \ell \le L - 1$.
4. *$\hat a^{[\ell_1]} f(\xi_{\ell_1}) \ne g(\xi_{\ell_1})\hat b^{[\ell_1]}$ for all ℓ_1 with $\deg(f) + \deg(g) + k \le \ell_1 \le L - 1$ and*
   $\hat a^{[\ell_2]} \bar f(\xi_{\ell_2}) \ne \bar g(\xi_{\ell_2})\hat b^{[\ell_2]}$ for all ℓ_2 with $\deg(f) + \deg(g) \le \ell_2 \le \deg(f) + \deg(g) + k - 1$.

Proof. Recall the system we solve is given by equation (4.12) and we solve $\hat a^{[\ell]}\Phi(\xi_\ell) = \Psi(\xi_\ell)\hat b^{[\ell]}$. Let

$$\Phi(u) = y_d u^d + y_{d-1}u^{d-1} + \ldots + y_0 \text{ and}$$
$$\Psi(u) = u^e + z_{e-1}u^{e-1} + \ldots + z_0,$$

where $d = \deg(f) + k$ and $e = \deg(g) + k$. For all ℓ such that $0 \le \ell \le \deg(f) + \deg(g) - 1$ let $\hat{a}^{[\ell]} = g(\xi_\ell)$ and $\hat{b}^{[\ell]} = f(\xi_\ell)$. Assume first $k = 0$, i.e., there are no errors. We set up and solve the non-homogeneous linear system

$$\hat{a}^{[\ell]}\Phi(\xi_\ell) - \Psi^*(\xi_\ell)\hat{b}^{[\ell]} = \hat{b}^{[\ell]}\xi_\ell^e, \tag{2.19}$$

where $\Psi^* = z_{e-1}u^{e-1} + z_{e-2}u^{e-2} + \ldots + z_0$.

Let $B\begin{bmatrix} y \\ z^* \end{bmatrix} = v$ be the matrix representation of our system in (2.19). We have for the right side vector v that $v \neq 0$ since $\hat{b}^{[\ell]} = f(\xi_\ell)$ cannot be zero for all $0 \le \ell \le \deg(f) + \deg(g) - 1$ since $\deg(g) \ge 1$ and $\xi_\ell \neq 0$ for all $0 \le \ell \le L - 1$. Our system then has L equations and $L + 1$ unknowns, so $B \in \mathsf{K}^{L\times(L+1)}$. By construction $\begin{bmatrix} f \\ g^* \end{bmatrix}$ is a solution to our system. Since our system is underdetermined there must be other solutions

$$\begin{bmatrix} \bar{f}_c \\ \bar{g}_c^* \end{bmatrix} = \begin{bmatrix} f \\ g^* \end{bmatrix} + cw$$

where $w \neq 0$ is in the null space of B and $c \neq 0$. Let $p = \mathrm{res}_u(f + cw_f, g + cw_{g^*})$, p is a polynomial in $c, p \neq 0$ since $p(0) \neq 0$. Note $\deg(p) \le \deg(f) + \deg(g)$ and $|\mathsf{K}| \ge 2(\deg(f) + \deg(g) + k) + 1$. Thus there must be $c_1 \in \mathsf{K}$ such that $c_1 \neq 0, p(c_1) \neq 0$ and $\mathrm{lc}(f) \neq -\mathrm{lc}(c_1 w_f)$. Consider $\bar{f} = \bar{f}_{c_1}$ and $\bar{g} = \bar{g}_{c_1}$. Then by construction $\deg(f) = \deg(\bar{f})$ and $\deg(g) = \deg(\bar{g})$. Also since $p(c_1) \neq 0$ we have that $\mathrm{GCD}(\bar{f}, \bar{g}) = 1$.

Next we show that $f/g \neq \bar{f}/\bar{g}$. We show first that $\begin{bmatrix} f \\ g^* \end{bmatrix}$ and $\begin{bmatrix} \bar{f} \\ \bar{g}^* \end{bmatrix}$ are linearly independent. Assume $\begin{bmatrix} f \\ g^* \end{bmatrix}$ and $\begin{bmatrix} \bar{f} \\ \bar{g}^* \end{bmatrix}$ are linearly dependent, then there exits $\alpha \neq 0$ such that $\alpha\begin{bmatrix} f \\ g^* \end{bmatrix} = \begin{bmatrix} \bar{f} \\ \bar{g}^* \end{bmatrix}$, which implies $\alpha\begin{bmatrix} f \\ g^* \end{bmatrix} = \begin{bmatrix} f \\ g^* \end{bmatrix} + c_1 w$, which further implies $(\alpha - 1)\begin{bmatrix} f \\ g^* \end{bmatrix} = c_1 w, \alpha \neq 1$ since $c_1 \neq 0$ and $w \neq 0$. So $\frac{\alpha-1}{c_1}\begin{bmatrix} f \\ g^* \end{bmatrix} = w$, but $0 \neq \frac{\alpha-1}{c_1}v = \frac{\alpha-1}{c_1}B\begin{bmatrix} f \\ g^* \end{bmatrix} = Bw = 0$, which is a contradiction. Thus $\begin{bmatrix} f \\ g^* \end{bmatrix}$ and $\begin{bmatrix} \bar{f} \\ \bar{g}^* \end{bmatrix}$ are linearly independent, which implies that $\begin{bmatrix} f \\ g \end{bmatrix}, \begin{bmatrix} \bar{f} \\ \bar{g} \end{bmatrix}$ are linearly independent. Which further implies that $f/g \neq \bar{f}/\bar{g}$.

To see why $\bar{g}(\xi_\ell) \neq 0$ for all ℓ with $0 \le \ell \le \deg(f) + \deg(g) - 1$, assume $\bar{g}(\xi_\ell) = 0$ for all ℓ with $0 \le \ell \le \deg(f) + \deg(g) - 1$. Since $\mathrm{GCD}(\bar{f}, \bar{g}) = 1$ and $\hat{a}^{[\ell]}\bar{f}(\xi_\ell) = \bar{g}(\xi_\ell)\hat{b}^{[\ell]}$ then $\bar{g}(\xi_\ell) = 0$ implies that $\hat{a}^{[\ell]} = 0$. This is a contradiction since $\hat{a}^{[\ell]} = g(\xi_\ell) \neq 0$ for all ℓ with

$0 \le \ell \le \deg(f)+\deg(g)-1$. Thus $\bar{g}(\xi_\ell) \neq 0$ for all ℓ with $0 \le \ell \le \deg(f)+\deg(g)-1$.

Now assume $k > 0$. By construction for all ℓ with $0 \le \ell \le \deg(f)+\deg(g)-1$ we have $\hat{a}^{[\ell]} f(\xi_\ell) - g(\xi_\ell)\hat{b}^{[\ell]} = 0$ and $\hat{a}^{[\ell]}\bar{f}(\xi_\ell) - \bar{g}(\xi_\ell)\hat{b}^{[\ell]} = 0$. Thus $\hat{a}^{[\ell]}(\bar{g}(\xi_\ell)f(\xi_\ell) - g(\xi_\ell)\bar{f}(\xi_\ell)) = 0$. Since $\hat{a}^{[\ell]} \neq 0$ it must be that $\bar{g}(\xi_\ell)f(\xi_\ell) - g(\xi_\ell)\bar{f}(\xi_\ell) = 0$. Since $f/g \neq \bar{f}/\bar{g}$, and $\mathrm{GCD}(f,g) = \mathrm{GCD}(\bar{f},\bar{g}) = 1$ then $\bar{g}f - g\bar{f} \in \mathsf{K}[u]$ is not identically zero. Since $\deg(f) = \deg(\bar{f})$ and $\deg(g) = \deg(\bar{g})$ and $(\bar{g}f - g\bar{f})(\xi_\ell) = 0$ for all ℓ with $0 \le \ell \le \deg(f)+\deg(g)-1$ we must have that $\deg(\bar{g}f - g\bar{f}) = \deg(f)+\deg(g)$. Observe that ξ_ℓ for $0 \le \ell \le \deg(f)+\deg(g)-1$ are $\deg(f)+\deg(g)$ distinct roots of $(\bar{g}f - g\bar{f})(u)$, so $(\bar{g}f - g\bar{f})(u)$ can have no other roots. Let $\hat{a}^{[\ell]} = g(\xi_\ell)$ and $\hat{b}^{[\ell]} = f(\xi_\ell)$ for all ℓ with $\deg(f)+\deg(g) \le \ell \le \deg(f)+\deg(g)+k-1$. Then for all ℓ with $0 \le \ell \le \deg(f)+\deg(g)+k-1$ we have $\hat{a}^{[\ell]} = g(\xi_\ell)$ and $\hat{b}^{[\ell]} = f(\xi_\ell)$ and therefore $\hat{a}^{[\ell]}f(\xi_\ell) - g(\xi_\ell)\hat{b}^{[\ell]} = 0$. By construction $\hat{a}^{[\ell]}\bar{f}(\xi_\ell) - \bar{g}(\xi_\ell)\hat{b}^{[\ell]} = 0$ for all ℓ with $0 \le \ell \le \deg(f)+\deg(g)-1$. Let $\hat{a}^{[\ell]} = \bar{g}(\xi_\ell)$ and $\hat{b}^{[\ell]} = \bar{f}(\xi_\ell)$ for all ℓ with $\deg(f)+\deg(g)+k \le \ell \le L-1$ then have we have $\hat{a}^{[\ell]}\bar{f}(\xi_\ell) - \bar{g}(\xi_\ell)\hat{b}^{[\ell]} = 0$ for all ℓ with $\deg(f)+\deg(g)+k \le \ell \le L-1$.

Assume there exist ξ_ℓ for some ℓ with $\deg(f)+\deg(g) \le \ell \le \deg(f)+\deg(g)+k-1$ such that $\hat{a}^{[\ell]}\bar{f}(\xi_\ell) - \bar{g}(\xi_\ell)\hat{b}^{[\ell]} = 0$. Then $(\bar{g}f - g\bar{f})(\xi_\ell) = 0$ for that ξ_ℓ. Which is a contradiction since we have already shown that if ξ_ℓ is a root of $(\bar{g}f - g\bar{f})(u)$ then $\ell < \deg(f)+\deg(g)$. Thus for all ℓ with $\deg(f)+\deg(g) \le \ell \le \deg(f)+\deg(g)+k-1$ we must have $\hat{a}^{[\ell]}\bar{f}(\xi_\ell) \neq \bar{g}(\xi_\ell)\hat{b}^{[\ell]}$. A similar argument shows that for all ℓ with $\deg(f)+\deg(g)+k \le \ell \le L-1$ we have $\hat{a}^{[\ell]}f(\xi_\ell) \neq g(\xi_\ell)\hat{b}^{[\ell]}$. Thus $\hat{a}^{[\ell_1]}f(\xi_{\ell_1}) \neq g(\xi_{\ell_1})\hat{b}^{[\ell_1]}$ for all ℓ_1 with $\deg(f)+\deg(g)+k \le \ell_1 \le L-1$ and $\hat{a}^{[\ell_2]}\bar{f}(\xi_{\ell_2}) \neq \bar{g}(\xi_{\ell_2})\hat{b}^{[\ell_2]}$ for all ℓ_2 with $\deg(f)+\deg(g) \le \ell_2 \le \deg(f)+\deg(g)+k-1$. □

We now show that if the solution $\frac{1}{g}f$ is such that $f^{[i_1]} = f^{[i_2]} \neq 0$ for all $1 \le i_1 < i_2 \le n$ then $\deg(g) \le \deg(A)$. Thus, by Theorem 3, if our parameters are exact we have that $L_{\mathrm{BK}} \le L_{\mathrm{CAB}}$.

Lemma 2 *If A is full rank, and the vector f has the property that $f^{[i_1]} = f^{[i_2]} \neq 0$ for all $1 \le i_1 < i_2 \le n$, and $Af = gb$ then $b \neq 0^m$.*

Proof. A full rank implies $\mathrm{rank}(A(u)) = n$. Assume $b = 0^m$, this implies $f^{[1]}\sum_{j=1}^n a_{i,j} = 0, i = 1,\dots,m$. Since $f \neq 0$ this is equivalent to $\sum_{j=1}^n A_j = 0$, which implies the columns of A are linearly dependent. Thus A is not full rank, which is a contradiction. □

Corollary 1 *If A is full rank and $f^{[i_1]} = f^{[i_2]} \neq 0$ for all $1 \le i_1 < i_2 \le n$ then $\deg(g) \le \deg(A)$, thus by Theorem 3 in the exact case $L_{\mathrm{BK}} \le L_{\mathrm{CAB}}$.*

Proof. Let A full rank and $A(\frac{1}{g}f) = b, g \neq 0$. This implies $Af = gb$, which further implies $f^{[1]}\sum_j a_{i,j} = gb_i$ for all i. We know by Lemma 2 that $b_i \neq 0$ for all i. Recall that if $\frac{1}{g}f$ is the solution to $Ax = b$ then $\mathrm{GCD}(f,g) = 1$. Thus $f^{[1]}\sum_j a_{i,j} = gb_i$ implies g divides $\sum_j a_{i,j}$ for all i. For those i such that $b_i \neq 0$, $\deg(g) \le \deg\left(\sum a_{i,j}\right) \le \max_{1\le i\le m, 1\le j\le n}\deg(a_{i,j}) = \deg(A)$. Thus $\deg(g) \le \deg(A)$. □

We now have two counts that we can use to solve the problem we describe in Remark 1. Theorem 3 tells us that whenever $\deg(g) > \deg(A)$ then the L_{CAB} count uses fewer evaluations than the L_{BK} count if all parameter values are exact. Lemma 1 shows however, that if $n = m = 1$ we cannot do better than the L_{BK} count. Lemma 2 and Corollary 1 tell us that if the solution $\frac{1}{g}f$ is such that $f^{[i_1]} = f^{[i_2]}$ for all $1 \leq i_1 < i_2 \leq n$ then it must be the case that the $\deg(A) > \deg(g)$. In the following section we combine the two counts to get a general early termination strategy. Such a termination strategy would be useful when little is known about the degree of the system and/or solution, since in such cases it is likely that the bounds one chooses are much larger than the actual value of the parameters.

2.5 Combined Early Termination

We now describe an algorithm that combines the early termination strategy for the L_{BK} count with early termination strategy for the L_{CAB} count. This strategy can be implemented when we are unsure how the $\deg(g)$ compares to the $\deg(A)$ and we suspect that our degree bounds significantly overestimates the actual values of their respective parameters.

Algorithm 4: Early Termination with L^*_{BK} and L^*_{CAB}

```
Input: $d_f \geq \deg(f), d_g \geq \deg(g), d_A \geq \deg(A), d_b \geq \deg(b)$
    $\rho_E < 1/2$, a rational number with denominator $q_E$,
            # the error rate
    $\rho_R < 1 - 2\rho_E$, a rational number with denominator $q_R$,
            # the rank drop rate, see Remark 2.
Output: $f, g$, and $\Lambda$.
 1: $d^*_f \leftarrow 0$; $d^*_g \leftarrow 0$.
 2: $D \leftarrow \min\big\{ \max\{d_f + d^*_g, d_g + d^*_f\}, \max\{d_A + d^*_f, d_b + d^*_g\}\big\} + 1$.
 3: $E^* \leftarrow \lfloor \bar{E}^* \rfloor; R^* \leftarrow \lfloor \bar{R}^* \rfloor$ where $\bar{E}^*$ and $\bar{R}^*$ are as defined in equations (2.16) and (2.17)
    respectively.
 4: if $\max\{d_f + d^*_g, d_g + d^*_f\} \leq \max\{d_A + d^*_f, d_b + d^*_g\}$ then
 5:   if Algorithm 1$(d_f, d_g, d^*_f, d^*_g, E^*, R^*)$
        returns at Step 9 then return $(f, g, \Lambda)$; end if
    else
 6:   if Algorithm 3$(d_A, d_b, d^*_f, d^*_g, E^*, R^*)$
        returns at Step 9 then return $(f, g, \Lambda)$; end if
    end if
 7: while(true) $D \leftarrow D + 1$.
 8:   Reassign $E^*, R^*$ as in Step 3 using the updated $D$ in equations
      (2.16) and (2.17) respectively.
 9:   for all $(d^*_f, d^*_g)$ with $D = \min\big\{ \max\{d_f + d^*_g, d_g + d^*_f\},$
                                 $\max\{d_A + d^*_f, d_b + d^*_g\}\big\} + 1$ do
10:     if $D = \max\{d_f + d^*_g, d_g + d^*_f\}$ then
11:       if Algorithm 1$(d_f, d_g, d^*_f, d^*_g, E^*, R^*)$
            returns at Step 9 then return $(f, g, \Lambda)$; end if
        else
12:       if Algorithm 3$(d_A, d_b, d^*_f, d^*_g, E^*, R^*)$
            returns at Step 9 then return $(f, g, \Lambda)$; end if
    end if; end for; end while
```

Remark 7 The justification for Algorithm 4 follows from the justification for Algorithm 2. If values for d_A and d_b are not known they can be set to infinity and Algorithm 4 becomes Algorithm 2. Similarly if values for d_f and d_g are not known they can be set to infinity and Algorithm 4 is the Cabay early termination algorithm.

2.6 Summary

[Boyer and Kaltofen 2014] give an algorithm that can be used to solve a black box parametric linear system in one variable. The algorithm works even if for some queries to the black box return erroneous results. Their algorithm uses the generalized Welch/Berlekamp count. The generalized Welch/Berlekamp count requires the sum of an upper bound for the common denominator and an upper bound for the corresponding numerators of the solution in order to determine the number of queries to the black box that are necessary to recover the solution. Thus, if one or both of the degree bounds are much larger than the actual degrees then the algorithm will make that much more queries to the black box, which in addition may cause an increase in the number of expected errors and places where the scalar matrices are rank deficient. Though their algorithm does not tell us precisely what happens if the degree bounds are not met, we showed that the algorithm can always diagnose if the degrees that are input to the algorithm do not bound the actual degrees of the solution. With this property, we can start with any degrees on input and increment them if the algorithm reports that they are too low. The algorithm terminates once the input degrees bound the actual degrees of the solution. If there is an error rate and/or a rank drop rate then the upper bounds for the number of errors and/or number of rank drops can be adjusted respectively each time the input degrees are incremented. This may cause further reduction of the number of queries to the black box.

We then showed how to compute another count for the number of queries to the black box that can be used to recover the solution even if for some queries the black box return erroneous results. The second count, based on work by [Cabay 1971] and later [Olesh and Storjohann 2007], though it requires an upper bound for the common denominator as well as an upper bound for the corresponding numerators it does not require their sum. Instead, it also requires knowledge of an upper bound for degree of the polynomials in the matrix as well as an upper bound for the degree of the polynomials in the right side vector. Like with the generalized Welch/Berlekamp count the algorithm can diagnose if the input degrees for the denominator and numerator do not bound from above the actual degrees of the solution. Thus the early termination strategy applies for Cabay count as well. We also showed that if our bounds are exact then the Cabay count is smaller than the generalized Welch/Berlekamp count only if the degree of the matrix is smaller than the degree of the common denominator. Thus if the degree of the solution is high relative to the degree of the system then the Cabay count would require fewer evaluations than the generalized Welch/Berlekamp count to recover the solution.

Finally we presented an algorithm that uses both bounds so as to achieve early termination regardless of the relationship between the degree of the system and its solution.

Chapter 3

Rational Vector Recovery with Error Correction as a Specialization of Parametric Linear System Solving

3.1 Introduction

Rational function vector recovery with errors is a special case of the algorithm in [Boyer and Kaltofen 2014] for parametric linear system solving with errors. If we consider $\hat{b}^{[\ell]} = \frac{1}{g(\xi_\ell)} f(\xi_\ell)$ and $\hat{A}^{[\ell]} = I_n$ then we can recover the rational function vector $\frac{1}{g(u)} f(u)$ from its evaluations, when some evaluations are erroneous, using the [Boyer and Kaltofen 2014] algorithm. Thus we can apply our early termination algorithms to the problem of rational function vector recovery with errors. There is just one caveat; for rational functions $\frac{1}{g(u)} f(u)$ where the $\deg(g) > \deg(A)$ we need more information at poles (when ξ_ℓ is a root of g). There are examples where it is not enough to just indicate that an evaluation point is a pole when attempting early termination. If we are to recover the rational function vector when some evaluations are poles then we need the black box to provide information about the numerators of the solution. We discuss in detail the additional information we require from the black box when it indicates that an evaluation is a pole.

3.2 Rational Vector Recovery

Suppose that there is a vector of rational functions $\frac{1}{g}f$ we wish to recover, and assume that this vector of rational functions is the unique solution to a system of linear equations

$$A(u)\,x = b(u), \quad A(u) \in \mathsf{K}[u]^{m\times n}, b(u) \in \mathsf{K}[u]^m,$$

where K is a field. See (2.10).
Let

$$\gamma_i^{[\ell]} = \begin{cases} f^{[i]}(\xi_\ell)/g(\xi_\ell) & \text{if } g(\xi_\ell) \neq 0 \\ \infty & \text{if } g(\xi_\ell) = 0. \end{cases}$$

We further assume that we have a black box that takes $\xi_\ell \in \mathsf{K}$ as inputs and returns vectors $\beta_i^{[\ell]}$ such that $\beta_i^{[\ell]} = \gamma_i^{[\ell]}$ for $\ell \notin \{\lambda_1, \ldots, \lambda_k\}$ and all $1 \le i \le n$ and $\beta_i^{[\ell]} \neq \gamma_i^{[\ell]}$ for $\ell \in \{\lambda_1, \ldots, \lambda_k\}$ on at least one i, $1 \le i \le n$. The remaining $m-n$ entries of the vector is filled with zeros. We show that using the model in [Boyer and Kaltofen 2014] as defined in Section 2.2, one can recover the rational vector $\frac{1}{g}f$. Recall that in the model $\hat{A}^{[\ell]}$ and $\hat{b}^{[\ell]}$ do not necessarily equal $A(\xi_\ell)$ or $b(\xi_\ell)$ respectively. We only need on sufficiently many evaluations to have $\hat{A}^{[\ell]}f(\xi_\ell) = g(\xi_\ell)\hat{b}^{[\ell]}$, and $\operatorname{rank}(\hat{A}^{[\ell]}) = n$.
Thus if we let

$$\hat{A}^{[\ell]} = \begin{bmatrix} I_n \\ 0 \ldots 0 \\ \vdots \ddots \vdots \\ 0 \ldots 0 \end{bmatrix} \qquad \text{and} \qquad \hat{b}^{[\ell]} = \begin{bmatrix} \beta_1^{[\ell]} \\ \vdots \\ \beta_n^{[\ell]} \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \tag{3.1}$$

for all ξ_ℓ such that $g(\xi_\ell) \neq 0$, and

$$\hat{A}^{[\ell]} = 0^{m\times n} \qquad \text{and} \qquad \hat{b}^{[\ell]} = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \tag{3.2}$$

whenever $g(\xi_\ell) = 0$ we can recover the vector of rational functions. We shall call the ξ_ℓ's such that $g(\xi_\ell) = 0$ the poles of the rational function. If none of our black box evaluations indicate that we have evaluated at a pole then $\hat{A}^{[\ell]}$ is always full rank. We know that we can recover the rational vector with $L = d_f + d_g + 2E + R + 1$ and $L = \max\{d_f + d_A, d_g + d_b\} + 2E + R + 1$ evaluations respectively. Note $R = 0$ since $\operatorname{rank}(\hat{A}^{[\ell]}) = n$ for all $0 \le \ell \le L - 1$. Now there must be a matrix $A(u)$ of minimal degree for which the vector $\frac{1}{g}f$ is the solution of the system $A(u)x = b(u)$. We have proved in Theorem 3 that in the cases where the $\deg(g) > \deg(A)$, $L = \max\{\deg(f) + \deg(A), \deg(g) + \deg(b)\} + 2k + 1 < L = \deg(f) + \deg(g) + 2k + 1$ so we can achieve Cabay early termination.

Suppose that on some evaluations of ξ_ℓ's the black box indicates, by the value ∞, that we have encountered a pole. We show that the count $L = d_f + d_g + 2E + 1$ evaluations suffices to recover the rational function vector. Ideally, we would like to say that this follows directly from Theorem 2, however we cannot guarantee that we have $\operatorname{rank}(\hat{A}^{[\ell]}) = n$ on $\ge d_f + d_g + E + 1$ many points for which $\hat{A}^{[\ell]} f(\xi_\ell) = g(\xi_\ell)\hat{b}^{[\ell]}$, one of the assumptions of Theorem 2. This full rank assumption is used in the proof of Theorem 2 only to establish that the vector of field elements $\Psi(\xi_\ell)f(\xi_\ell) - g(\xi_\ell)\Phi(\xi_\ell) = 0$. Thus if we can establish that the vector of field elements $\Psi(\xi_\ell)f(\xi_\ell) - g(\xi_\ell)\Phi(\xi_\ell) = 0$ without using the fact that the $\operatorname{rank}(\hat{A}^{[\ell]}) = n$ on $\ge d_f + d_g + E + 1$ many points we would establish our claim as all other assumptions of Theorem 2 remain the same.

Proof. There are two possibilities on non-erroneous evaluations of ξ_ℓ, that is $\ell \notin \{\lambda_1, \dots, \lambda_k\}$:

1. $g(\xi_\ell) = 0$ which implies $\Psi(\xi_\ell) = 0$. See (3.2).
2. $g(\xi_\ell) \neq 0$ which implies $\Phi(\xi_\ell) = \Psi(\xi_\ell)\frac{1}{g(\xi_\ell)} f(\xi_\ell)$.

Recall that we solve equation (4.12). Note that in both cases we indeed have the vector of field elements $\Psi(\xi_\ell)f(\xi_\ell) - g(\xi_\ell)\Phi(\xi_\ell) = 0$. Thus our claim is established. $\square$

Remark 8 The system formed by using $\hat{A}^{[\ell]}$ and $\hat{b}^{[\ell]}$ as described in (3.1) and (3.2) above is overdetermined. We show in Lemma 1 that without additional information about the errors we can always construct a second solution that has the same characteristics as the actual solution. Nevertheless, given appropriate assumptions about the error locations one can reduce the number of necessary equations. For instance, in decoding interleaved Reed-Solomon codes it is assumed that the errors occur in bursts, that is errors occur in blocks [Bleichenbacher, Kiayias, and Yung 2003; Schmidt, Sidorenko, and Bossert 2009]. In a forthcoming paper we will give the analysis for a semi-deterministic scenario, that is where the actual errors do not need to be random field elements. Note that Theorem 3 describes a second scenario where the number of evaluations is less for interleaved codes, namely when the vector encodes a rational function that is the solution to a parametric linear system (see also [Pernet 2014]). $\square$

3.3 Cabay Early Termination with poles

Suppose $\deg(g) > \deg(A)$ and that on evaluation at some ξ_ℓ's, $\ell \notin \{\lambda_1, \ldots, \lambda_k\}$, the black box indicates that $g(\xi_\ell)$ is a pole. There are examples where $L_{\text{CAB}} = \max\{d_f + d_A, d_g + d_b\} + 2E + 1$ evaluations are not sufficient to recover the rational function vector using our current model for rational vector recovery. To prove that L_{CAB} was sufficient to recover the rational function vector $\frac{1}{g}f$ we needed the $\text{rank}(\hat{A}^{[\ell]}) = n$ for all ξ_ℓ, $\ell \notin \{\lambda_1, \ldots, \lambda_k\}$. We needed this to establish that $A(\xi_\ell)\Phi(\xi_\ell) = \Psi(\xi_\ell)b(\xi_\ell)$ and the pair $(\Lambda f, \Lambda g)$ is a solution to our linear system, where $\Lambda(u)$ is the error locator polynomial. However in our rational vector recovery model we set $\hat{A}^{[\ell]} = 0$ whenever $g(\xi_\ell) = 0$ for all ℓ, see (3.2). Note that in the current rational vector recovery model when $g(\xi_\ell) = 0$ we set $\Psi(\xi_\ell) = 0$ and lose all information about $\Phi(\xi_\ell)$, see (3.2). Consequently we may not be able to recover $\frac{1}{g}f$ as we may not have enough information about f. To remedy the lack of information we adjust our black box output to gain some information about f at poles. Let

$$\gamma_i^{[\ell]} = \begin{cases} \frac{1}{g(\xi_\ell)} f(\xi_\ell) & \text{if } g(\xi_\ell) \neq 0 \\ \left.\begin{array}{l} w^{[1]}, \ldots, w^{[r_\ell]}, \text{ a basis for the} \\ \text{null space of } A(\xi_\ell), \\ \qquad\qquad \textbf{\textit{or}} \\ c_\ell f(\xi_\ell), \text{ a non-zero scalar multiple} \\ \text{of the evaluated numerator vector,} \\ \text{both with an indication that } g(\xi_\ell) = 0 \end{array}\right\} & \text{if } g(\xi_\ell) = 0 \end{cases}$$

be what the black box returns. We show that if at the poles we add the equations $\Phi(\xi_\ell) = \Theta_{\ell,1}w^{[1]} + \ldots + \Theta_{\ell,r_\ell}w^{[r_\ell]}$, or $\Phi(\xi_\ell) = \Theta_\ell c_\ell f(\xi_\ell), c_\ell \neq 0$, to the set of equations produced by the original rational vector recovery model then we can recover $\frac{1}{g}f$ with $L_{\text{CAB}} = \max\{d_f + d_A, d_g + d_b\} + 2E + 1$ evaluations, where $\Theta_j \in \mathsf{K}$ for all j are new unknowns.

Theorem 4 *Suppose that for $\geq \max\{d_f + d_A, d_g + d_b\} + E + 1$, ξ_ℓ we have $\beta_i^{[\ell]} = \gamma_i^{[\ell]}$ for all i. If we add*

$$\Psi(\xi_\ell) = 0 \qquad \textit{and} \tag{3.3}$$

$$\Phi(\xi_\ell) = \Theta_{\ell,1}w^{[1]} + \ldots + \Theta_{\ell,r_\ell}w^{[r_\ell]} \ \textit{or} \tag{3.4}$$

$$\Phi(\xi_\ell) = \Theta_\ell c_\ell f(\xi_\ell), c_\ell \neq 0. \tag{3.5}$$

to the system we solve, whenever $\gamma_i^{[\ell]} = \infty$ for all $i, 1 \leq i \leq m$, and if $\Phi \in \mathsf{K}[u]^n, \Psi \in \mathsf{K}[u]$, and $\Theta_{\ell,1}, \ldots, \Theta_{\ell,r_\ell} \in \mathsf{K}$, or $\Theta_\ell \in \mathsf{K}$ is a solution of the system, then for the pair (Φ, Ψ) that we compute we have $A(\xi_\ell)\Phi(\xi_\ell) = \Psi(\xi_\ell)b(\xi_\ell)$, and $(\Lambda f, \Lambda g)$ solve (3.3)*, and* (3.4) *or* (3.5)*.*

Proof. Note that the black box can return $w^{[1]}, \ldots, w^{[r_\ell]}$ for some poles and $c_\ell f(\xi_\ell), c_\ell \neq 0$ for others. If $g(\xi_\ell) = 0$ then we add two sets of equations, (3.3), and (3.4) or (3.5). Clearly $\Psi(\xi_\ell)b(\xi_\ell) = 0^m$, and for (3.4) we have

$$\begin{aligned} A(\xi_\ell)\Phi(\xi_\ell) &= A(\xi_\ell)(\Theta_{\ell,1}w^{[1]} + \ldots + \Theta_{\ell,r_\ell}w^{[r_\ell]}) \\ &= \Theta_{\ell,1}A(\xi_\ell)w^{[1]} + \ldots + \Theta_{\ell,r_\ell}A(\xi_\ell)w^{[r_\ell]} = 0^m, \end{aligned}$$

or for (3.5) we have

$$\begin{aligned} A(\xi_\ell)\Phi(\xi_\ell) &= A(\xi_\ell)(\Theta_\ell c_\ell f(\xi_\ell)) \\ &= \Theta_\ell c_\ell A(\xi_\ell)f(\xi_\ell) = g(\xi_\ell)b(\xi_\ell) = 0^m. \end{aligned}$$

Thus we indeed have $A(\xi_\ell)\Phi(\xi_\ell) = \Psi(\xi_\ell)b(\xi_\ell)$ whenever $g(\xi_\ell) = 0$. Consider $\Lambda(\xi_\ell)A(\xi_\ell)f(\xi_\ell) = \Lambda(\xi_\ell)g(\xi_\ell)b(\xi_\ell) = 0^m$. We always have $A(\xi_\ell)f(\xi_\ell) = g(\xi_\ell)b(\xi_\ell) = 0^m$ when $g(\xi_\ell) = 0$. This implies that $f(\xi_\ell)$ must be in the null space of $A(\xi_\ell)$. Thus $f(\xi_\ell) = \sum_j d_{\ell,j}w^{[j]}, d_{\ell,j} \in \mathsf{K}$. So if at a pole we add equation (3.4), then $\Lambda(\xi_\ell)f(\xi_\ell) = \Lambda(\xi_\ell)\sum_j d_{\ell,j}w^{[j]}$, so $\Theta_{\ell,j} = \Lambda(\xi_\ell)d_{\ell,j}$ implies that $\Lambda(\xi_\ell)f(\xi_\ell)$ solves (3.4). If we add (3.5) at a pole, observe that $\Lambda(\xi_\ell)f(\xi_\ell) = \Theta_\ell c_\ell f(\xi_\ell)$ implies $\Theta_\ell = \Lambda(\xi_\ell)/c_\ell$. So $\Lambda(\xi_\ell)f(\xi_\ell)$ solves (3.5). Clearly $\Lambda(\xi_\ell)g(\xi_\ell)$ is a solution to (3.3). □

3.4 Summary

The recovery of a black box vector rational functions from its evaluation when some evaluations maybe erroneous is a specialization of the algorithm for full rank parametric linear system solving with error correction. In chapter 2 we discussed early termination strategies for full rank parametric linear system algorithm. We showed in this chapter that rational vector recovery with errors is a special case of the problem of finding a solution to parametric system when some evaluations are erroneous. Since rational vector recovery is just a special case of the parametric linear system problem our early termination strategies apply. If we allow some evaluations of the rational function vector to be poles (roots of the common denominator) then some additional information is required if the Cabay count is used. See section 3.3 for a full discussion of the problem that poles present as well as the additional information we require to recover the rational function vector.

Chapter 4

Parametric Linear System Solving with Error Correction for Rank Deficient Systems

4.1 Introduction

Consider consistent linear systems of the form $A(u)x = b(u)$, where $A(u) \in \mathsf{K}[u]^{m\times n}$, $m \geq n$ and $\operatorname{rank}(A(u)) = r < n$. That is, $A(u)$ is rank deficient. The right side vector $b(u) \in \mathsf{K}[u]^m$ and K is a field. Such systems have as their solutions rational function vectors $x(u)$ with entries $x_i(u) = f^{[i]}(u)/g^{[i]}(u)$, $1 \leq i \leq n$. We consider solutions, $\frac{1}{g(u)}f(u)$, to be such that $g(u)$ is the monic least common denominator, that is

$$\mathrm{GCD}(f, g) \overset{\text{def}}{=} \mathrm{GCD}(\mathrm{GCD}_i(f^{[i]}), g) = 1.$$

Since the matrices of the systems we consider are rank deficient a solution that has $g(u)$ as the denominator is not unique like in the full rank case discussed in Chapter 2. See also [Boyer and Kaltofen 2014]. As in the full rank case, however, the solutions of such systems can be determined by evaluating the system at distinct points, ξ_ℓ, from the field K and interpolating the evaluated solution [McClellan 1973]. Like the full rank case a degree bounded solution can be found even if some evaluations are erroneous. Once again for a non-erroneous evaluation point, ξ_ℓ, it is not necessary to have evaluations of the matrix, $A(\xi_\ell)$, and right side vector, $b(\xi_\ell)$, in order to interpolate the solution. We require only a scalar matrix, $\hat{A}^{[\ell]} \in \mathsf{K}^{m\times n}$, and right side vector, $\hat{b}^{[\ell]} \in \mathsf{K}^m$, that agrees with any evaluated solution. That is for all $x \in \mathsf{K}^n$ such that $A(\xi_\ell)x = b(\xi_\ell)$ we have $\hat{A}^{[\ell]}x = \hat{b}^{[\ell]}$. Though a polynomial matrix is rank deficient it has a fixed rank, and like in the full rank case there are finitely many evaluations which may cause the evaluated matrix to drop below the true rank.

Consider the following model. Suppose there exists an oracle, which we will refer to as the black box. If we supply the black box with a value, ξ_ℓ, from the field K the black box returns to us $\hat{A}^{[\ell]} \in \mathsf{K}^{m \times n}$ and $\hat{b}^{[\ell]} \in \mathsf{K}^m$. The scalar matrix, $\hat{A}^{[\ell]}$, and right side vector, $\hat{b}^{[\ell]}$, that are returned may not be $A(\xi_\ell)$ and $b(\xi_\ell)$. Nevertheless, if we query the black box L times we assume that $\leq E$ times we get a scalar matrix, $\hat{A}^{[\lambda]}$, and right side vector $\hat{b}^{[\lambda]}$ such that there exists a solution of the evaluated system that is not a solution to the scalar system returned by the black box. That is $\leq E$ times the black box returns a scalar system such that there exists x such that $A(\xi_\lambda)x = b(\xi_\lambda)$, but $\hat{A}^{[\lambda]}x \neq \hat{b}^{[\lambda]}$. Such evaluations are considered to be erroneous. Furthermore, we assume that $\leq R$ times the black box returns a scalar matrix, $\hat{A}^{[\ell]}$, and right side vector, $\hat{b}^{[\ell]}$, such that there are no errors but the $\operatorname{rank}(\hat{A}^{[\ell]}) < r$. The objective is to find a degree bounded solution, $x(u) = \frac{1}{g(u)} f(u)$, of the system $A(u)x(u) = b(u)$. The degree bounds for the solution are known a-priori and are inputs to the algorithm. Observe that this model is a generalization of the model for full rank systems.

Unlike in the full rank case our algorithm does not simultaneously compute the error locator polynomial along with a degree bounded solution. Our algorithm instead identifies and removes the errors before computing the solution. In order to identify the errors we separate all possible errors into two categories. The first category of errors we refer to as Matrix Errors. Matrix Errors occur if the scalar matrix, $\hat{A}^{[\lambda]}$, returned by the black box is such that there is no scalar right vector that would make the scalar system consistent for all solutions of the evaluated system. See Definition 2. A key observation is that Matrix Errors occur at an evaluation, ξ_λ, if and only if the null space of the evaluated matrix, $A(\xi_\lambda)$, is not equal to the null space of the scalar matrix, $\hat{A}^{[\lambda]}$. Thus, we identify and remove all Matrix Errors by essentially interpolating the null space of $A(u)$ from the scalar matrices returned by the black box.

The second category of errors we refer to as Right Side Vector Errors. Right Side Vector Errors occur if the scalar matrix, $\hat{A}^{[\lambda]}$, returned by the black box is such that there exists a scalar right side vector, $\tilde{b}$, for which the scalar system in not erroneous, however the scalar right side vector, $\hat{b}^{[\lambda]}$, returned by the black box is not equal to $\tilde{b}$. See Definition 6. Notice that because the solution is unique in the full rank case all errors are Right Side Vector Errors. Thus to locate right side vector errors we identify linearly independent columns of the scalar systems and run the full rank algorithm. Recall that the full rank algorithm computes an error locator polynomial and solution simultaneously. We may not be able to use the solution computed because it may not be bounded by our input degree bounds. Nevertheless, we are able to identify all Ride Side Vector Errors.

Since any error is either a Matrix Error or a Right Side Vector Error we can identify and remove all errors. Thus allowing us to solve an error free system. The solutions to the error free system are bounded by the input degree bounds.

4.2 Exact Vector-of-Functions Solving - Rank Deficient Systems

We solve a system of linear equations

$$A(u)\,x(u) = b(u), \quad A(u) \in \mathsf{K}[u]^{m\times n}, b(u) \in \mathsf{K}[u]^m, \tag{4.1}$$

where K is a field and $m \geq n$. We shall assume that the system is rank deficient, but consistent. Any solution $x(u)$ of the system can be expressed in the form

$$x(u) = \begin{bmatrix} \vdots \\ \frac{1}{g(u)} f^{[i]}(u) \\ \vdots \end{bmatrix} \in \mathsf{K}(u)^n, \quad g \neq 0, \tag{4.2}$$

where g is a least common (monic) denominator, whose leading coefficient in u is 1.

Unlike the full rank case discussed in [Boyer and Kaltofen 2014] and [Kaltofen, Pernet, Storjohann, and Waddell 2017], a solution with particular degree in g is not unique. So we compute one solution from the set of degree-bounded solutions namely,

$$\frac{1}{g} f \in S_{d_f,d_g}\{(y,z) \mid Ay = zb, y \in \mathsf{K}[u]^m, \forall\, i : \deg(y_i) \leq d_f, z \in \mathsf{K}[u], z \neq 0, \deg(z) \leq d_g\}$$

Definition 1 *Let $g_{\min}$ be a monic least common denominator with minimal degree.*

Theorem 5 *If $\frac{1}{g_{\min}} f$ is a solution to equation (4.1) then $g_{\min}$ is unique.*

Proof. Assume $\frac{1}{g_1} f$ and $\frac{1}{g_2} \bar{f}$ are solutions of our linear system such that g_1 and g_2 are monic, of minimal degree, and $g_1 \neq g_2$. Then we have $Af = g_1 b$ and $A\bar{f} = g_2 b$. So that $Af - A\bar{f} = g_1 b - g_2 b$. We then have that $A(f - \bar{f}) = (g_1 - g_2)b$. Let $f - \bar{f} = f^*$ and $g_1 - g_2 = g^* \neq 0$, then $Af^* = g^* b$ implies $\frac{1}{g^*} f^*$ is another solution to our linear system such that $\deg(g^*) < \deg(g_1) = \deg(g_2)$. This contradicts the assumption that $\deg(g_1) = \deg(g_2)$ is minimal. Thus $g_{\min}$ is unique. □

Since the matrices of the systems we are solving are rank deficient a solution with minimal degree in the denominator is not unique among all solutions. Nevertheless, by theorem 5 we know that the monic denominator of minimal degree is unique among all solutions. Once again we do not have access to the matrix $A(u)$ and right side vector $b(u)$. We do, however, have access to a black box. The black box when supplied with a value, ξ_ℓ, from the field K returns a scalar matrix $\hat{A}^{[\ell]} \in \mathsf{K}^{m\times n}$ and scalar right side vector $\hat{b}^{[\ell]} \in \mathsf{K}^m$. The scalar matrix $\hat{A}^{[\ell]}$ and right side vector $\hat{b}^{[\ell]}$ are not necessarily $A(\xi_\ell)$ and $b(\xi_\ell)$ respectively.

Let $W_\ell = \{w \in \mathsf{K}^n \mid A(\xi_\ell)w = b(\xi_\ell)\}$ and $\widehat{W}_\ell = \{\hat{w} \in \mathsf{K}^n \mid \hat{A}^{[\ell]}\hat{w} = \hat{b}^{[\ell]}\}$. There is an error at ξ_λ if there exists $w \in W_\lambda$ such that $w \notin \widehat{W}_\lambda$. That is $W_\lambda \not\subseteq \widehat{W}_\lambda$. If we query the black box L times then errors occur at $\leq E$ many evaluations.

The matrix $A(u)$ has a definitive rank $r < n$. Like in the full rank case there are finitely many evaluations that may cause the rank to drop below r. Nonetheless, with high probability the rank of $A(\xi_\ell) = r$, [Storjohann and Villard 2005]. We assume that $\leq R$ many queries, ξ_ℓ, to the black box returns $\hat{A}^{[\ell]}$ and $\hat{b}^{[\ell]}$ such that no error occurs but the rank$(\hat{A}^{[\ell]}) < r$.

Theorem 6 *If the* rank$(A(u)) = r$ *and* $g_{\min}(\xi_\ell) = 0$ *then the* rank$(A(\xi_\ell)) < r$.

Proof. The rank$(A(u)) = r$ implies that A has at least one $r \times r$ non singular submatrix. Let q_r be the determinant of any such a submatrix, $q_r \in \mathsf{K}[u]$ and $q_r \neq 0$. By Cramer's rule there always exist solutions such that the common denominator g is a factor of some q_r. Let $\frac{1}{g_r}f_r$ be a solution where g_r is a factor of some q_r. Note that the deg$(g_r) \geq$ deg$(g_{\min})$. Let c_r be the quotient when we divide g_r by $g_{\min}$. Observe that $\frac{1}{c_r g_{\min}}c_r f$ is a solution to our system. So we have $Af_r = g_r b$ and $Ac_r f = c_r g_{\min} b$. This implies that $A(f_r - c_r f) = (g_r - c_r g_{\min})b$. Now $g_r - c_r g_{\min}$ is the remainder when g_r is divided by $g_{\min}$, thus deg$(g_r - c_r g_{\min}) <$ deg$(g_{\min})$. This would then imply that $\frac{1}{g_r - c_r g_{\min}}(f_r - c_r f)$ is a solution to our system that has denominator degree less than $g_{\min}$, which is a contradiction. Thus $g_r - c_r g_{\min} = 0$ and $g_{\min} | g_r$. This further implies that $g_{\min} | q_r$. Since $g_{\min}(\xi_\ell) = 0$ then it must be that $q_r(\xi_\ell) = 0$, so rank$(A(\xi_\ell)) < r$. $\square$

It should be possible to compute a solution vector $\frac{1}{g_{\min}}f$ from scalar matrices, $\hat{A}^{[\ell]}$, and right side vectors $\hat{b}^{[\ell]}$ where $\hat{A}^{[\ell]}$ and $\hat{b}^{[\ell]}$ are not necessarily $A(\xi_\ell)$ and $b(\xi_\ell)$ respectively as long as errors occur at $\leq E$ evaluations. We should also be able to compute an error locator polynomial, Λ, without having to check the solution against all the systems returned by the black box. One idea is to use the full rank algorithm with the rank deficient interpretation of E and R. That is, let

$$L_{\text{CAB}} = \max\{d_f + d_A, d_g + d_b\} + 2E + R + 1 \tag{4.3}$$

where

$$E \geq |\{\lambda \mid W_\lambda \not\subseteq \widehat{W}_\lambda \text{ for } 0 \leq \lambda \leq L_{\text{CAB}} - 1\}|, \tag{4.4}$$

and

$$R \geq |\{\ell \mid W_\ell \subset \widehat{W}_\ell \text{ for } 0 \leq \ell \leq L_{\text{CAB}} - 1\}|. \tag{4.5}$$

The bounds d_f, d_g, R and E are input to the algorithm. The required number of evaluations are named for [Cabay 1971], see also [Olesh and Storjohann 2007]. Recall that we solve the linear system

$$\hat{A}^{[\ell]} \begin{bmatrix} \vdots \\ \Phi^{[i]}(\xi_\ell) \\ \vdots \end{bmatrix} - \Psi(\xi_\ell)\hat{b}^{[\ell]} = 0, \quad \deg(\Phi^{[i]}) \le d_f + E, \deg(\Psi) \le d_g + E, 0 \le \ell \le L_{\text{CAB}} - 1. \quad (4.6)$$

in the unknown coefficients $\Phi^{[i]}(u)$ and $\Psi(u)$. The linear system (4.6) has $n(d_f+E+1)+d_g+E+1$ unknown coefficients for $\Phi^{[i]}(u)$ and $\Psi(u)$ and mL_{CAB} equations.

Lemma 3 *If the* $\operatorname{rank}(A(\xi_\ell)) = \operatorname{rank}(\hat{A}^{[\ell]})$ *and* $W_\ell \subseteq \widehat{W}_\ell$ *then* $W_\ell = \widehat{W}_\ell$.

Proof. The general solution of any nonhomogeneous linear system consists of a particular solution for the nonhomogeneous system plus the general solution of the corresponding homogeneous system. We know that the solutions to the homogeneous systems are exactly the entries in the null space of the matrix. While the null space of a matrix is a vector space, the set of all solutions of a nonhomogeneous linear system is not a vector space. Instead the set of all solutions to a nonhomogeneous linear system forms an affine subspace that is a translation of the null space. The dimension of the affine space is equal to the dimension of the translated space. So for linear systems the dimension of the space containing all solutions is equal to the dimension of the null space of the matrix. Thus the $\dim(W_\ell) = \dim(N(A(\xi_\ell)))$. Since we assume $\operatorname{rank}(A(\xi_\ell)) = \operatorname{rank}(\hat{A}^{[\ell]})$ then by the Rank Plus Nullity Theorem we know that the $\dim(N(A(\xi_\ell))) = \dim(N(\hat{A}^{[\ell]}))$. Since the $\dim(N(\hat{A}^{[\ell]})) = \dim(\widehat{W}_\ell)$ we have that the $\dim(W_\ell) = \dim(\widehat{W}_\ell)$. Since $W_\ell \subseteq \widehat{W}_\ell$ it must be that $W_\ell = \widehat{W}_\ell$. □

Lemma 4 *If* $W_\ell = \widehat{W}_\ell$ *then the* $N(A(\xi_\ell)) = N(\hat{A}^{[\ell]})$.

Proof. Consider $\hat{v} \in N(\hat{A}^{[\ell]})$, then $\hat{w} + \hat{v} \in \widehat{W}_\ell$ and thus $\hat{w} + \hat{v} \in W_\ell$. So $A(\xi_\ell)(\hat{w} + \hat{v}) = b(\xi_\ell)$. Now $\hat{w} \in \widehat{W}_\ell$ implies that $\hat{w} \in W_\ell$ so $A(\xi_\ell)(\hat{w} + \hat{v} - \hat{w}) = b(\xi_\ell) - b(\xi_\ell) = 0$, which implies $\hat{v} \in N(A(\xi_\ell))$. Thus the $N(\hat{A}^{[\ell]}) = N(A(\xi_\ell))$. □

Theorem 7 *Suppose that for* $\le E$ *of the* ξ_ℓ *we have* $W_\ell \ne \widehat{W}_\ell$ *and for* $\ge \max\{d_f + d_A, d_g + d_b\} + E + 1$ *of the* ξ_ℓ *we have* $W_\ell = \widehat{W}_\ell$ *then* $(\Lambda f, \Lambda g_{\min})$ *is in the set of solutions to (4.6).*

Proof. Consider $\Psi(\xi_\ell) \ne 0$ we have $\hat{A}^{[\ell]}\Phi(\xi_\ell) = \Psi(\xi_\ell)\hat{b}^{[\ell]}$ implies that $\hat{A}^{[\ell]}(\Phi(\xi_\ell)/\Psi(\xi_\ell)) = \hat{b}^{[\ell]}$ which implies $\Phi(\xi_\ell)/\Psi(\xi_\ell) \in \widehat{W}_\ell$. Thus $\Phi(\xi_\ell)/\Psi(\xi_\ell) \in W_\ell$ which implies $A(\xi_\ell)(\Phi(\xi_\ell)/\Psi(\xi_\ell)) = b(\xi_\ell)$ which further implies $A(\xi_\ell)\Phi(\xi_\ell) = \Psi(\xi_\ell)b(\xi_\ell)$. Now for $\Psi(\xi_\ell) = 0$ we have $\hat{A}^{[\ell]}\Phi(\xi_\ell) = 0$ which implies that $\Phi(\xi_\ell) \in N(\hat{A}^{[\ell]})$. In Lemma 4 we show that the $N(A(\xi_\ell)) = N(\hat{A}^{[\ell]})$. So if $\hat{A}^{[\ell]}\Phi(\xi_\ell) = 0$ then $A(\xi_\ell)\Phi(\xi_\ell) = 0$, so for all ξ_ℓ such that $\ell \notin \{\lambda_1, \ldots, \lambda_k\}$, $\hat{A}^{[\ell]}\Phi(\xi_\ell) = \Psi(\xi_\ell)\hat{b}^{[\ell]}$ implies $A(\xi_\ell)\Phi(\xi_\ell) = \Psi(\xi_\ell)b(\xi_\ell)$. Since $A(u)\Phi(u) - \Psi(u)b(u)$ is a polynomial vector of degree

$\leq \max\{d_f + d_A, d_g, d_b\} + E$ that vanishes at $\max\{d_f + d_A, d_g + d_b\} + E + 1$ distinct points we have $A(u)\Phi(u) = \Psi(u)b(u)$. Observe that the pair $(\Phi, \Psi) = (\Lambda f, \Lambda g_{\min})$ solves (4.6) where $\Lambda(u)$ is the error locator polynomial. Thus $(\Lambda f, \Lambda g_{\min})$ is in the space of solutions to (4.6). □

Theorem 7 tells us that solutions of the type we are after are contained in the set of solutions to equation (4.6). We compute a basis for the solution space of equation (4.6), however it is difficult to extract $(\Lambda f, \Lambda g_{\min})$. Unlike the full rank case, $\Psi_{\min}$ is not necessarily $\Lambda g_{\min}$.We know that if there are no errors then we can recover a degree bounded solution $\frac{1}{g_{\min}} f$. Thus the strategy we employ is to exploit the properties and structure of rank deficient matrices in order to locate and remove all erroneous evaluations. Once all erroneous evaluations are removed we can utilize the algorithm from [Boyer and Kaltofen 2014], see also [Kaltofen, Pernet, Storjohann, and Waddell 2017], with the, L_{CAB}, to recover the degree bounded solution. That is we solve the linear system

$$\hat{A}^{[\ell]} \begin{bmatrix} \vdots \\ \Phi^{[i]}(\xi_\ell) \\ \vdots \end{bmatrix} - \Psi(\xi_\ell)\hat{b}^{[\ell]} = 0, \quad \deg(\Phi^{[i]}) \leq d_f, \deg(\Psi) \leq d_g, 0 \leq \ell \leq L_{\text{CAB}} - 1. \tag{4.7}$$

in the unknown coefficients $\Phi^{[i]}(u)$ and $\Psi(u)$. The linear system (4.7) has $n(d_f + 1) + d_g + 1$ unknown coefficients for $\Phi^{[i]}(u)$ and $\Psi(u)$ and mL_{CAB} equations. Observe that any solution to equation (4.7) is in S_{d_f,d_g}. That is the solutions are bounded by the input degree bounds. Thus once all errors are removed me can compute a degree bounded solution.

Errors that occur are one of two types. We shall refer to these two types of errors as Matrix Errors and Right Side Vector Errors. We will discuss these two types of errors and how they are removed in the two sections that follow.

4.3 Removing Matrix Error

Recall that $W_\ell = \{w \in \mathsf{K}^n \mid A(\xi_\ell)w = b(\xi_\ell)\}$ and $\widehat{W}_\ell = \{\hat{w} \in \mathsf{K}^n \mid \hat{A}^{[\ell]}\hat{w} = \hat{b}^{[\ell]}\}$. Also recall that there is an error at ξ_λ if there exists $w \in W_\lambda$ such that $w \notin \widehat{W}_\lambda$. That is, $W_\lambda \not\subseteq \widehat{W}_\lambda$.

Definition 2 ***Matrix Error:*** *For all $\tilde{b} \in \mathsf{K}^n$ there exists $w \in W_\lambda$ such that $\hat{A}^{[\lambda]}w \neq \tilde{b}$.*

Theorem 8 *There is no matrix error at ξ_ℓ if and only if $N(A(\xi_\ell)) \subseteq N(\hat{A}^{[\ell]})$.*

Proof. If there is no matrix error then $N(A(\xi_\ell)) \subseteq N(\hat{A}^{[\ell]})$.
If there is no matrix error we know that there exists $\tilde{b} \in \mathsf{K}^n$ such that for all $w \in W_\ell$ we have $\hat{A}^{[\ell]}w = \tilde{b}$. Assume that $\hat{b}^{[\ell]} = \tilde{b}$. Then $W_\ell \subseteq \widehat{W}_\ell$. So if $w \in W_\ell$ then $w \in \widehat{W}_\ell$. Let $v \in N(A(\xi_\ell))$ then $w + v \in W_\ell$ and $w + v \in \widehat{W}_\ell$. Now consider

$$\begin{aligned}\hat{A}^{[\ell]}(w + v - w) &= \hat{b}^{[\ell]} - \hat{b}^{[\ell]} \\ &= 0.\end{aligned}$$

Thus $v \in N(\hat{A}^{[\ell]})$. So if there is no matrix error then $N(A(\xi_\ell)) \subseteq N(\hat{A}^{[\ell]})$. □

Proof. If $N(A(\xi_\ell)) \subseteq N(\hat{A}^{[\ell]})$ then there is no matrix error.

Let

$$\{v_1^{[\ell]}, v_2^{[\ell]}, \ldots, v_\rho^{[\ell]}\}, \text{ where } \rho = n - \operatorname{rank}(A(\xi_\ell)),$$

be a basis for the $N(A(\xi_\ell))$. Then for all w such that $A(\xi_\ell)w = b(\xi_\ell)$,

$$w = x_0 + \sum_{i=1}^{\rho} \alpha_i v_i^{[\ell]} \text{ where } \alpha_i \in \mathsf{K} \text{ and } x_0 \text{ is such that } A(\xi_\ell)x_0 = b(\xi_\ell).$$

Then

$$\begin{aligned}\hat{A}^{[\ell]}w &= \hat{A}^{[\ell]}\left(x_0 + \sum_{i=1}^{\rho} \alpha_i v_i^{[\ell]}\right) \\ &= \hat{A}^{[\ell]}x_0 + \sum_{i=1}^{\rho} \alpha_i \hat{A}^{[\ell]} v_i^{[\ell]} \\ &= \hat{A}^{[\ell]}x_0\end{aligned}$$

since $N(A(\xi_\ell)) \subseteq N(\hat{A}^{[\ell]})$. Thus if $N(A(\xi_\ell)) \subseteq N(\hat{A}^{[\ell]})$ then there exists $\tilde{b} = \hat{A}^{[\ell]}x_0$ such that for all w such that $A(\xi_\ell)w = b(\xi_\ell)$ we have that $\hat{A}^{[\ell]}w = \tilde{b}$. Therefore, if $N(A(\xi_\ell)) \subseteq N(\hat{A}^{[\ell]})$ then there is no matrix error. □

From Theorem 8 we know that if the black box returned $\hat{A}^{[\lambda]}$ and $\hat{b}^{[\lambda]}$, the $\operatorname{rank}(\hat{A}^{[\lambda]}) = \operatorname{rank}(A(\xi_\lambda))$ and it is the case that a Matrix Error occurred, then it must be that the $N(A(\xi_\lambda)) \neq$

$N(\hat{A}^{[\lambda]})$. Since $A(u) \in \mathsf{K}[u]^{m\times n}$ and the $\operatorname{rank}(A(u)) = r < n$ we must have that $\operatorname{rank}(A(\xi_\lambda)) \le r$. So $N(A(\xi_\lambda)) \ne \varnothing$. We then can remove Matrix Errors by identifying all evaluations, ξ_λ, such that the $N(A(\xi_\lambda)) \ne N(\hat{A}^{[\lambda]})$. We shall now describe our method for removing matrix errors.

Definition 3 ***Non Essential Column:*** *Let $A{\downarrow}_{*,j}$ be the matrix that results if column j is removed from A. The column $A_{*,j}$ is a Non Essential Column of A if and only if* $\operatorname{colsp}(A) = \operatorname{colsp}(A{\downarrow}_{*,j})$.

Definition 4 ***Minimally Linearly Dependent Set:*** *A set of vectors, S, is a Minimally Linearly Dependent Set of vectors if and only if S is a linearly dependent set and no proper subset of S is a linearly dependent set.*

Definition 5 ***Right Most Minimally Linearly Dependent Set of Columns:*** *Let $M \in \mathsf{K}^{m\times n}$ such that* $\operatorname{rank}(M) < n$. *Beginning with the right most column, ignoring any zero columns, and proceeding left one column at a time we add the columns to a set. Each time we add a column to the set we check whether the set of columns are linearly dependent. If the set has become linearly dependent we stop adding columns and find the minimal linear dependent subset. The vectors in the minimal linear dependent subset we refer to as the Right Most Minimally Linearly Dependent Set of Columns of M.*

Example 1

$$M = \begin{bmatrix} 3 & 2 & 1 & 4 \\ 4 & 2 & 1 & 5 \\ 5 & 2 & 1 & 6 \\ 6 & 2 & 1 & 7 \end{bmatrix}$$

The set $\left\{ \begin{matrix} 2 \\ 2 \\ 2 \\ 2 \end{matrix}, \begin{matrix} 1 \\ 1 \\ 1 \\ 1 \end{matrix} \right\}$ *is the Right Most Minimally Linearly Dependent Set of Columns of M.*

Lemma 5 *The non zero entries in the last column of a column echelon form for a basis for the $N(M)$ corresponds to the Right Most Minimally Linearly Dependent Set of Columns of M.*

Proof. The $N(M)$ must contain a vector that only has non zero entries that correspond to the Right Most Minimally Linearly Dependent Set of Columns of M. Observe that by definition this vector will have the maximum number of zero entries above the first non-zero entry in the vector. Recall that the last column of a column echelon form for a basis of a vector space has the following shape:

$$\begin{bmatrix} * & 0 & 0 & \dots & 0 \\ * & * & 0 & \dots & 0 \\ * & * & * & \dots & 0 \\ \vdots & \vdots & \vdots & & \vdots \\ * & * & * & \dots & 0 \\ * & * & * & \dots & * \\ * & * & * & \dots & * \\ \vdots & \vdots & \vdots & & \vdots \\ * & * & * & \dots & * \end{bmatrix}$$

Note that the last vector in the column echelon form for the basis of a vector space has the maximum zero entries before the first non zero entry in the vector. We know that all null space vectors are a linear combination of the basis vectors. Observe that this implies that the last column in a column echelon form for a basis of a vector space always has the same number of zeros above the first non zero entry. Furthermore, this number of zeros is greater than or equal to all other counts for the number of zeros that precede the first non zero entry for all null space vectors. Thus the non zero entries in the last column of a column echelon form for a basis for the $N(M)$ corresponds to the Right Most Minimally Linearly Dependent Set of Columns of M. □

Remark 9 Observe that in Example 1 the matrix M is square of dimension 4. The $\text{rank}(M) = 2$, so that $n - r = 4 - 2 = 2$. If we solve the equation $M\Omega = 0$ then a matrix Ω that contains a basis for the solution in column echelon form is

$$\Omega = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & -2 \\ -1 & 0 \end{bmatrix}.$$

Notice that the non zero entries in $\Omega_{*,2}$ correspond to the Right Most Minimally Linearly Dependent Set of Columns (see definition 5) of the matrix M in Example 1. □

Lemma 6 *Let $M \in K^{m\times n}$ such that* $\text{rank}(M) < n$.
Let $M_{*,j}$ be a Non Essential Column of M.
There exists a permutation matrix, P, such that $M_{*,j}$ is in the Right Most Minimally Linearly Dependent Set of Columns of MP.

Proof. Since $M_{*,j}$ is a Non Essential Column of M we know that $M_{*,j}$ is in the span of the other columns of M. Consider a basis for the column space of $M{\downarrow_{*,j}}$. Observe that if that basis along with $M_{*,j}$ are the last columns of MP then they form the Right Most Minimally Linearly Dependent Set of Columns of MP. $\square$

From Lemma 6 we know that every Non Essential Column of $A(u)$ is a member of at least one Minimally Linearly Dependent subset of columns of $A(u)$. Consider a particular Minimally Linearly Dependent Set that is a subset of the columns of $A(u)$. We know that there are vectors in the $N(A(u))$ that are non zero only in the entries that correspond to the columns in the Minimally Linearly Dependent Set. For vectors in the null space that only are only non zero in the entries that correspond to the columns in the Minimally Linearly Dependent Set there must be a vector of minimal degree.

Let $\{\nu_1(u), \nu_2(u), \ldots, \nu_\eta(u)\}$ be a linearly independent set of columns in $\operatorname{colsp}(A{\downarrow_{*,j}}(u))$ such that

$$A_{*,j}(u) = \frac{1}{c_0(u)} \sum_{i=1}^{\eta} c_i(u)\nu_i(u).$$

where $A{\downarrow_{*,j}}$ is a Non Essential Column of $A(u)$.

$$d_j^{[non.ess.]} \overset{\text{def}}{=} \min_{\nu_1(u),\nu_2(u),\ldots,\nu_\eta(u)} (\max_{0\le i\le\eta} (\deg(c_i(u))).$$

$$d_\Omega \overset{\text{def}}{=} \max\{d_j^{[non.ess.]}\} \cup \{d_f\} \tag{4.8}$$

Let

$$L_{\text{SING}} = d_A + d_\Omega + R + 2E + 1.$$

We shall compute a degree bounded basis for the $N(A(u))$, with respect to K that has degree equal to $d_\Omega + E$.

Consider the homogeneous system of linear equations

$$A(u)y = 0^m,\ A(u) \in \mathsf{K}[u]^{m\times n},\ \operatorname{rank}(A(u)) = r < n \tag{4.9}$$

where y is a degree bounded vector in the $N(A(u))$.

We then solve

$$\hat{A}^{[\ell]} \begin{bmatrix} \vdots \\ \Omega^{[i]}(\xi_\ell) \\ \vdots \end{bmatrix} = 0^m, \quad \deg(\Omega^{[i]}) \le d_\Omega + E, 0 \le \ell \le L_{\text{SING}} - 1. \tag{4.10}$$

The homogeneous linear system (4.10) has $n(d_\Omega + E + 1)$ unknown coefficients for the $\Omega^{[i]}(u)$

and $m(d_A + d_\Omega + 2E + R + 1)$ many equations.

Theorem 9 *We suppose that $\leq E$ of the the evaluations, ξ_λ, are erroneous. Therefore for $\geq d_A + d_\Omega + E + 1$ of the evaluations, ξ_ℓ, the* $\text{rank}(\hat{A}^{[\ell]}) = r$ *and there are no errors. Then a basis for the solutions of Equation (4.10) forms a degree bounded subspace of the $N(A(u))$ over* K.

Proof. The solutions of equation (4.10) are vectors of the form:

$$\begin{bmatrix} \omega^{[1]}_{d_A+d_\Omega+E} \\ \omega^{[1]}_{d_A+d_\Omega+E-1} \\ \vdots \\ \omega^{[1]}_0 \\ \omega^{[2]}_{d_A+d_\Omega+E} \\ \omega^{[2]}_{d_A+d_\Omega+E-1} \\ \vdots \\ \omega^{[2]}_0 \\ \vdots \\ \omega^{[n]}_{d_A+d_\Omega+E} \\ \omega^{n}_{d_A+d_\Omega+E-1} \\ \vdots \\ \omega^{[n]}_0 \end{bmatrix}.$$

The component $\omega^{[i]}_j$ is the $u^{j^{th}}$ coefficient in the polynomial $\Omega^{[i]}(u)$. We have that for all evaluations, ξ_ℓ, such that $0 \leq \ell \leq d_A + d_\Omega + E$ we have that the $N(A(\xi_\ell)) = N(\hat{A}^{[\ell]})$. Take any solution vector for equation (4.10) converted to polynomial vector $\Omega(u)$ and compute the polynomial vector $A(u)\Omega(u)$. Observe that the $\deg(A(u)\Omega(u)) \leq d_A + d_\Omega + E$ but for at least $d_A + d_\Omega + E + 1$ evaluation, ξ_ℓ, we have that $\hat{A}^{[\ell]}\Omega(\xi_\ell) = 0^m$. Since for at least $d_A + d_\Omega + E + 1$ evaluation $N(\hat{A}^{[\ell]}) = N(A(\xi_\ell))$ we have that $A(\xi_\ell)\Omega(\xi_\ell) = 0^m$. This implies that $A(u)\Omega(u) = 0^m$. Thus for all solutions of Ω in equation (4.10) we have that $\Omega(u)$ is in the $N(A(u))$. Therefore a basis for the solutions of equation (4.10) forms a degree bounded subspace over K for the $N(A(u))$. $\square$

From a basis for the solutions of equation (4.10) we can create a matrix, where the columns are basis vectors for the solution of equation (4.10). We shall refer to this matrix as the Ω-Solution Space Matrix. We then can compute the column echelon form of the Ω-Solution Space Matrix. Since the columns in the Ω-Solution Space Matrix are linearly independent then the last column in the column echelon form is not the zero vector.

Let Ω_{last} be the last vector in the column echelon form of the Ω-Solution Space Matrix.

Lemma 7 *The nonzero polynomials in Ω_{last} correspond to the Last Set of Linearly Dependent Columns of $A(u)$.*

Proof. Again recall that the column echelon form for a matrix with linearly independent columns has the following shape

$$\begin{bmatrix} * & 0 & 0 & \dots & 0 \\ * & * & 0 & \dots & 0 \\ * & * & * & \dots & 0 \\ \vdots & \vdots & \vdots & & \vdots \\ * & * & * & \dots & 0 \\ * & * & * & \dots & * \\ * & * & * & \dots & * \\ \vdots & \vdots & \vdots & & \vdots \\ * & * & * & \dots & * \end{bmatrix}$$

The vectors, $\Omega(u)$, that we compute in equation (4.10) are in the right $N(A(u))$. This means that they represent linear combinations of the columns of $A(u)$ that produce the zero vector. Note that all Essential Columns of $A(u)$ will correspond to the zero polynomial in the $N(A(u))$, where as the Non Essential Columns may correspond to non zero polynomials in the $N(A(u))$. Consider the Right Most Minimally Linearly Dependent Set of Columns of $A(u)$. Recall that there exist a vector in the null space of $A(u)$ such that the only non zero entries correspond to the columns in the Right Most Minimally Linearly Dependent Set of Columns of $A(u)$ and d_Ω, see equation (4.8), is an upper bound on its degree. Therefore a null space vector that is non zero only in the entries that correspond to the columns in the Right Most Minimally Linearly Dependent Set of Columns of $A(u)$ is in the span of the Ω-Solution Space Matrix. Furthermore, among all vectors in the span of the Ω-Solution Space Matrix with the property that they are non zero only in the entries that correspond to the Right Most Minimally Linearly Dependent Set of Columns of $A(u)$; there must be a vector with minimal degree in the first polynomial. Observe that you cannot combine Ω_{last} with the other columns in the Ω-Solution Space Matrix with out making one of the zero entries above the first non zero entry in Ω_{last} non zero. Assume that the non zero polynomials in Ω_{last} do not correspond to the Right Most Minimally Linearly Dependent Set of Columns of $A(u)$. Then the null space vector contained in the column span of the Ω-Solution Space with the following properties:

1. The only non zero polynomials are the polynomials that correspond to the Right Most Minimally Linearly Dependent Set of Columns of $A(u)$

2. The degree of the polynomial that corresponds to the left most vector in the Right Most Minimally Linearly Dependent Set of Columns of $A(u)$ is minimal

cannot be found by a linear combination over K of the columns of the Ω-Solution Space Matrix. This is a contradiction. Hence, the non zero polynomials in Ω_{last} correspond to the Right Most Minimally Linearly Dependent Set of Columns of $A(u)$. $\square$

Theorem 10 *We suppose that for $\geq d_A + d_\Omega + E + 1$ of the evaluation points, ξ_ℓ, there are no errors. And that $\leq E$ of the evaluations, ξ_λ, are erroneous. We know that if some errors are Matrix Errors then one or more of the linear dependencies that exist among the columns in $A(\xi_\lambda)$ does not exist among the columns in $\hat{A}^{[\lambda]}$. See Definition 2 and Theorem 8. The linear dependence that occurs among the Right Most Minimally Linearly Dependent Set of Columns of $A(\xi_\lambda)$ does not exist among the Right Most Minimally Linearly Dependent Set of Columns of $\hat{A}^{[\lambda]}$ if and only if Ω_{last} has $(u - \xi_\lambda)$ as a factor.*

Proof. By Theorem 9 the degree bounded Ω-Solution Space Matrix is not the zero matrix. Consider, Ω_{last}, the last column in the column echelon form of the Ω-Solution Space Matrix. Note that Ω_{last} is non zero since the matrix contains linearly independent columns, because the columns form a basis for a degree bounded subspace for $N(A(u))$ with respect to K. By the shape of the column echelon form we know that Ω_{last} contains the minimum degree of the first non zero polynomial in the column. See Lemma 7. We show first that if there are common root, ξ_λ, of the polynomials in $\Omega_{\text{last}}(u)$ then a matrix error occurred.

Assume there is a common root, ξ_λ, in $\Omega_{\text{last}}(u)$, but a matrix error did not occur. Since a matrix error did not occur we know that the same column dependencies that exist $A(\xi_\lambda)$ also exists in $\hat{A}^{[\lambda]}$. We know, from Lemma 7, that the non zero entries in $\Omega_{\text{last}}(u)$ correspond to the Right Most Minimally Linearly Dependent Set of Columns of $A(u)$. Note that the Right Most Minimally Linearly Dependent Set of Columns of $A(u)$ always forms the Right Most Minimally Linearly Dependent Set of Columns in $A(\xi_\lambda)$ when the $\operatorname{rank}(A(\xi_\lambda)) = r$. Since the columns are linearly dependent there must exist a solution where all of the non zero polynomials in $\Omega_{\text{last}}(u)$ would evaluate to a non zero value at ξ_λ. Therefore, there is another solution such that for all evaluations, except the evaluation at ξ_λ, the new solution agrees with $\Omega_{\text{last}}(u)$. However, at ξ_λ the solution gives a non zero linear combination that makes the dependent set of columns in $A(\xi_\lambda)$ that correspond to the Right Most Minimally Linearly Dependent Set of Columns of $A(u)$ equal zero. Observe that by interpolation this solution has degree no more than the corresponding polynomials in $\Omega_{\text{last}}(u)$. But by Lemma 7 the new solution cannot be a linear combination of degree bounded subspace which is a contradiction. Thus if there is a common root among all polynomials in $\Omega_{\text{last}}(u)$ then a matrix error has occurred.

Next we show that if a matrix error occurs because the relationship among the Right Most Minimally Linearly Dependent Set of Columns of $A(\xi_\lambda)$ does not exist among that corresponding columns in $\hat{A}^{[\lambda]}$ then there is a common factor $u - \xi_\lambda$ in $\Omega_{\text{last}}(u)$. If a matrix error occurs at ξ_λ then the $N(A(\xi_\lambda)) \neq N(\hat{A}^{[\lambda]})$. So we know there is a difference in one or more column dependencies. Assume that the column dependency differs due to one or more of the columns that correspond to the Right Most Minimally Linearly Dependent Set of Columns of $A(u)$. Then the same non zero linear combination cannot cause the Right Most Minimally Dependent Set of Columns of $A(\xi_\lambda)$ and the Right Most Minimally Linear Dependent Set of Columns of $\hat{A}^{[\lambda]}$to be zero. We know we compute a degree bounded subspace for the $N(A(u))$. Therefore $A(\xi_\lambda)\Omega_{\text{last}}(\xi_\lambda) = 0$. By construction we always have that $\hat{A}^{[\lambda]}\Omega_{\text{last}}(\xi_\lambda) = 0$. Thus $\Omega_{\text{last}}(\xi_\lambda) = 0$. Therefore $u - \xi_\lambda$ is a common factor in Ω_{last}. □

Remark 10 Matrix errors occur if one or more of the column dependencies that exists among the Non Essential Columns in $A(\xi_\lambda)$ does not exist as a column dependency among the columns $\hat{A}^{[\lambda]}$. The proof of Theorem 10 suggests an algorithm for removing matrix errors that occur because a column dependency that exists among the Right Most Minimally Linearly Dependent Set of Columns of $A(u)$ does not exist among the corresponding columns of $\hat{A}^{[\lambda]}$. It is possible, however, that the column dependency that exists in $A(\xi_\lambda)$ but does not exist in $\hat{A}^{[\lambda]}$ occurs among linearly dependent sets of columns that do not correspond to the Right Most Minimally Linearly Dependent Set of Columns of $A(u)$. We thus need to find a way to remove matrix errors that occur among linearly dependent sets of columns that do not correspond to the Right Most Minimally Linearly Dependent Set Columns of $A(u)$. The idea we implement is to make every linearly dependent set of columns the Right Most Minimally Linearly Dependent Set of Columns. Recall that d_Ω is an upper bound on the degree of some polynomial vector that only has non zero entries in the columns that correspond to Minimally Linearly Dependent Sets of columns for all Minimally Linearly Dependent Sets of Columns of $A(u)$. See Equation (4.8). We know that column operations affect the null space of a matrix, however column interchanges only result in a corresponding interchange of the entries in the null space vector.

Example 2

$$\begin{bmatrix} 1 & 2 & 8 \\ 2 & 4 & 16 \\ 3 & 6 & 24 \end{bmatrix} \begin{bmatrix} 4 \\ 2 \\ -1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \qquad \begin{bmatrix} 2 & 8 & 1 \\ 4 & 16 & 2 \\ 6 & 24 & 3 \end{bmatrix} \begin{bmatrix} 2 \\ -1 \\ 4 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

Notice that the reordering of the columns resulted in the corresponding reordering of the entries in the null space vector. This is due to the fact that multiplying a matrix, A, by a vector, x, on the right results in a vector, b, such that the entries in b are a linear combination of the columns in the matrix, and the coefficients of the linear combination are the entries in x.

Example 3

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} e \\ f \end{bmatrix} = \begin{bmatrix} ae + bf \\ ce + df \end{bmatrix} = e \begin{bmatrix} a \\ c \end{bmatrix} + f \begin{bmatrix} b \\ d \end{bmatrix}$$

We apply this idea to solving equation (4.10). Notice that in the implementation we do not have to reorder the columns of the individual $\hat{A}^{[\ell]}$'s. We can simply reorder the Ω's in equation (4.10). What we now need is a method to locate all the different Minimally Linearly Dependent Sets of Columns. One idea would be to consider all combinations of two or more columns as the last set of columns. Note that for every check we will discover a Right Most Minimally Linearly Dependent Set of Columns. The Right Most Minimally Linearly Dependent Set of Columns we discover may not be a new set, therefore it would be a good idea to memoize so that we do not check a set that we already know is Minimally Linearly Dependent Set of Columns.

We currently do know in the worst case how many different Minimally Linearly Dependent Sets of columns a matrix with n columns and of rank $= r$ my contain. We conjecture that the number of Minimally Linearly Dependent Set of Columns of a matrix with n columns is polynomial in n. Which would imply that there is polynomial time algorithm for finding all Matrix Errors. Further more because the intersection of some Minimally Linearly Dependent Sets are non empty and that Minimally Linearly Dependent Sets of Columns only contain Non Essential Columns we conjecture that we do not need to check all possible Minimally Linearly Dependent Sets of Columns. The idea is that we can describe some Minimally Linearly Dependent Sets using a combination of other Minimally Linearly Dependent Sets. We describe an algorithm, without proof, that is polynomial in n and works for all our examples. □

4.3.1 The Algorithm

Algorithm 5: Compute Matrix Errors

```
Input: a stream $(\hat{A}^{[\ell]}, \hat{b}^{[\ell]})$, $\ell = 0, 1, \ldots, L_{\text{SING}}$  # the scalar matrices and right side vectors
    # returned from the black box
Output: $\bar{\Lambda}$  # a factor of $\Lambda$
 1: $\Xi \leftarrow \{\xi_0, \xi_2, \ldots, \xi_{L_{\text{SING}}}\}$
 2: count $\leftarrow 0$ # counts the number of different permutations of the columns that have
    # been checked
 3: $\bar{\Lambda} \leftarrow 1$
 4: Solve
    $$\hat{A}^{[\ell]}\Omega(\xi_\ell) = 0, \qquad (4.11)$$
    # for all $\xi_\ell \in \Xi$ and $\deg(\Omega) \leq d_\Omega + E$
 5: compute CEF($\Omega$)
 6: $\Omega_{\text{last}} \leftarrow$ last column in CEF($\Omega$)
 7: first_$\Omega_0(u) \leftarrow$ first nonzero polynomial in $\Omega_{\text{last}}$
 8: $\bar{\Lambda} \leftarrow \bar{\Lambda}$GCD($\Omega_{\text{last}}(u)$)
 9: if first entry in $\Omega_{\text{last}}(u)$ is not zero then  # this implies that there is only one dependence
    # relationship among the columns
10:     count $\leftarrow n$  # skip the while loop
11: end if
12: move first_$\Omega_0(u)$ to be first in $\Omega(u)$
13: while count $< n$ do
14:     compute CEF($\Omega$)
15:     $\Omega_{\text{last}} \leftarrow$ last column in CEF($\Omega$)
16:     $\Omega_0(u) \leftarrow$ first nonzero polynomial in $\Omega_{\text{last}}$
17:     $\bar{\Lambda} \leftarrow \bar{\Lambda}$GCD($\Omega_{\text{last}}(u)$)
18:     if first entry in $\Omega_{\text{last}}(u)$ is not zero then
19:         break
20:     end if
21:     move $\Omega_0(u)$ to be first in $\Omega(u)$
22:     count $\leftarrow$ count+1
23:     if first_$\Omega_0(u)$ is again $\Omega_0(u)$ then  # checked all minimally linearly dependent sets
    # that involve columns to the right of the columns that refer $\Omega_0(u)$
24:         $\Omega(u) \leftarrow \Omega(u)$ with the $\Omega(u)^{[i]}$'s to the right if $\Omega_0(u)$ removed
```

```
25:     **end if**
26: **end while**
27: **return** $\bar{\Lambda}$
```

Remark 11 Algorithm 5 takes as input a stream of scalar matrices, $\hat{A}^{[\ell]}$, and right side vectors $\hat{b}^{[\ell]}$, that are returned by the black box when queried with the ξ_ℓ's and outputs a polynomial $\bar{\Lambda}$ a factor of the error locator polynomial Λ. The roots of the polynomial $\bar{\Lambda}$ are the evaluations that cause matrix errors.

The algorithms begins by solving the homogeneous system $\hat{A}^{[\ell]}\Omega(\xi_\ell) = 0$, for all $\xi_\ell \in \Xi$ and $\deg(\Omega) \leq d_\Omega + E$. In order to diagnose if matrix errors occurred Algorithm 5 first computes the column echelon form of, Ω, the solution of the homogeneous system $\hat{A}^{[\ell]}\Omega(\xi_\ell) = 0$. We then examine the last column, Ω_{last}, in the column echelon form of Ω. By Theorem 10 we know that common roots among the polynomials in Ω_{last} are exactly the places were matrix error occur. Further recall that Ω_{last} corresponds to the Right Most Minimally Linearly Dependent Set of Columns in $A(u)$. See Lemma 7. To find other matrix errors not exposed by the Right Most Minimally Linearly Dependent Column of $A(u)$ we change the order of the polynomials in $\Omega(u)$. Observe that changing the order of the polynomials is essentially changing the order of the columns in $A(u)$. We change the order in such a way to force a different Minimally Linearly Dependent Set of Columns in $A(u)$ to be the Right Most Minimally Linearly Dependent Set of Columns of the matrix with the corresponding change in the column order of $A(u)$. Because changing the column order of a matrix only changes the order of the entries in the null space the common roots of the polynomials in the new Ω_{last} are also matrix errors.

To reorder the polynomials in Ω record the first nonzero polynomial in the initial Ω_{last} then move it to the first position in Ω. Next compute a new Ω_{last} and check whether the first nonzero polynomial is the polynomial we recorded. If it is we restart the process of checking for matrix errors using only the polynomial we stored initially and those to the left of it in Ω, otherwise the first nonzero polynomial of the new Ω_{last} is moved to the first position in Ω and we restart the process of checking for matrix error with all of Ω. We claim, without proof, that if the interchanges are done in the manner described then no more than $n-1$ interchanges are necessary for computing all matrix errors, though there may be more than $n-1$ Minimally Linearly Dependent Sets of Columns in $A(u)$. $\square$

Remark 12 Matrix errors are specific to the rank deficient case. There are only right side vector errors in the full rank case. In the full rank case the solution, $\frac{1}{g}f$, is unique. Thus for all full rank scalar matrices, $\hat{A}^{[\ell]}$ returned by the black box there is a right side vector, namely $\hat{b}^{[\ell]} = \hat{A}^{[\ell]}\frac{1}{g(\xi_\ell)}f(\xi_\ell)$, for which the solution works. $\square$

4.4 Removing Right Side Vector Errors

Definition 6 ***Right Side Vector Error:*** *There exists $w \in W_\lambda$ such that $w \notin \widehat{W}_\lambda$ for the given $\hat{A}^{[\lambda]}$ and $\hat{b}^{[\lambda]}$, but there exists a scalar right side vector such that there is no error.*

In the previous section we discussed how to identify evaluated matrices $\hat{A}^{[\lambda]}$ that have matrix errors. Once these scalar matrices have been identified they can be removed. In this section we discuss how to identify errors in the right side vector. Recall that right side vector errors are exactly the type of errors that affect full rank systems. See Remark 12. The algorithm in [Boyer and Kaltofen 2014], see also [Kaltofen, Pernet, Storjohann, and Waddell 2017], tell us how to identify these errors when the system is full rank.

We now solve the system

$$\Psi(\xi_\ell)\hat{b}^{[\ell]} - \hat{A}^{[\ell]}\Phi(\xi_\ell) = 0 \text{ for all } \xi_\ell \text{ that remain in } \Xi \tag{4.12}$$

where $\deg(\Psi) \leq d_g + E$ and $\deg(\Phi) \leq d_f + E$.

Since we have removed all matrix errors we know that the scalar matrices that remain are all row equivalent. Row operations preserve the linear relationships that exist among the columns of a matrix. Thus, the reduced row echelon form of row equivalent matrices are identical. This fact is proven in most linear algebra text books. If we perform the row operations on the augmented system the solution is preserved. This is essentially Gaussian Elimination. Another benefit of the row echelon form is that it makes it easy to identify the linearly independent columns and hence the linearly dependent columns. In the row echelon form the columns that contain the pivots are linearly independent. Since we can identify the columns that are with out pivots we can remove those columns along with the corresponding $\Phi^{[i]}$'s. We then solve a system that has linearly independent columns. From [Boyer and Kaltofen 2014] we know that the roots of the $\text{GCD}(\Psi_{\min}, \Phi_{\min})$ are errors in the right side vector. See also [Kaltofen, Pernet, Storjohann, and Waddell 2017]. The solution produced by this reduced system is not guaranteed to be bounded by our input degree bounds. Nevertheless, we are able to find the errors that remain among the scalar systems returned by the black box.

Note that we need not scan equation 4.12 and remove the $\Phi^{[i]}$'s that correspond to non pivot columns. Instead, in our implementation, see Appendix A, we add the equation $\Phi^{[i]}(\xi_\ell) = 0$ if column i does not have a pivot in the reduced row echelon form. We should also note that we do not compute the $\text{rank}(A(u))$ so we cannot just use the row echelon form of one scalar matrix returned by the black box in order to determine the pivot columns. If a random scalar matrix is chosen to determine the pivot columns without first computing the rank then the algorithm is Monte Carlo. Nevertheless, we can gain speed in our algorithm by checking the rank of $R + 1$ many scalar matrices and using a matrix with maximum rank to determine the column dependencies. Note that if $\text{rank}(\hat{A}^{[\lambda]}) > r$ then a matrix error occurred by the rank plus nullity theorem. Thus any $\hat{A}^{[\lambda]}$ such that $\text{rank}(\hat{A}^{[\lambda]}) > r$ is removed. Another consequence of not computing the rank a-priori is that is possible that on some evaluations the pivot columns along with the right side vector may evaluate to zero, causing a common root in $\Phi_{\min}$ and $\Psi_{\min}$ which would indicate a false error. To correct this we remove one such linear factor from both $\Phi_{\min}$ and $\Psi_{\min}$ before computing the error.

4.4.1 The Algorithm

Algorithm 6: Compute Right Side Vector Errors

```
Input: $d_f \geq \deg(f), d_g \geq \deg(g_{\min})$, $d_A \geq \deg(A(u))$,
    a stream $(\hat{A}^{[\ell]}, \hat{b}^{[\ell]})$,  # the ones that were not found to be erroneous by
    # Algorithm 5
    $\bar{\Lambda}$  # returned by Algorithm 5
    $R \geq \left|\{\ell \mid \hat{A}^{[\ell]} f(\xi_\ell) = g(\xi_\ell)\hat{b}^{[\ell]} \text{ and } \text{rank}(\hat{A}^{[\ell]}) < r, 0 \leq \ell \leq L_{\text{CAB}} - 1\}\right|$,
    $E \geq \left|\{\lambda \mid \hat{A}^{[\lambda]} f(\xi_\lambda) \neq g_{\min}(\xi_\lambda)\hat{b}^{[\lambda]}, 0 \leq \lambda \leq L_{\text{CAB}} - 1\}\right|$
Output: $\Lambda$
 1: $E \leftarrow E - \deg(\bar{\Lambda})$  # $\bar{\Lambda}$ was computed in Algorithm 5
 2: $L_{\text{CAB}} \leftarrow \max\{d_A + d_f, d_b + d_g\} + R + 2E + 1$
 3: Find all dependent columns of the $\text{REF}(\hat{A}^{[\ell]})$ for all $\xi_\ell$ remaining in $\Xi$
 4: Solve simultaneously:

        $\Psi(\xi_\ell)\hat{b}^{[\ell]} - \hat{A}^{[\ell]}\Phi(\xi_\ell) = 0$ for all $\xi_\ell \in \Xi$
        and
        $\Phi^{[j]}(\xi_\ell) = 0$

    # for $L_{\text{CAB}}$ many $\xi_\ell \in \Xi$ where $\deg(\Phi) \leq d_f + E$ and $\deg(\Psi) \leq d_g + E$
    # and where column $j$ is a pivot less column in $\text{REF}(\hat{A}^{[\ell]})$
 5: for the $L_{\text{CAB}}$ many $\xi_\ell \in \Xi$ do
 6:   if $\hat{b}^{[\ell]}$ is the zero vector and $\hat{A}^{[\ell]}$ contains zero columns then
 7:     if $u - \xi_\ell \mid \Psi_{\min}(u)$ and $\Phi_{\min}(u)^{[j]}$ for all $j$ such that $j$ is a zero column in $\hat{A}^{[\ell]}$ then
    # $\xi_\ell$ is a non erroneous factor of the $\text{GCD}(\Psi_{\min}, \Phi_{\min})$
    # so remove one factor of $u - \xi_\ell$
 8:       Remove a factor of $u - \xi_\ell$ from $\Psi_{\min}(u)$ and $\Phi_{\min}(u)^{[j]}$
 9:     end if
10:   end if
```

```
11: end loop
12: $\Lambda \leftarrow \bar{\Lambda} \cdot \mathrm{GCD}(\Psi_{\min}(u), \Phi_{\min}(u))$
13: return $\Lambda$
```

4.5 Returning a degree bounded solution in the Rank Deficient Case

In Section 4.3 we describe how to detect and thus remove all systems containing matrix errors from the set of scalar systems returned by the black box. Then in Section 4.4 we describe how to identify and then remove all systems that contain right side vector errors. Observe that all possible errors belong to either matrix errors or right side vector errors. Thus, using the ideas presented in Sections 4.3 and 4.4 we can remove all erroneous evaluations and then solve

$$\Psi(\xi_\ell)\hat{b}^{[\ell]} - \hat{A}^{[\ell]}\Phi(\xi_\ell) = 0 \text{ for all } \xi_\ell \text{ that remain in } \Xi, \tag{4.13}$$

where $\deg(\Psi) \le d_g$ and $\deg(\Phi) \le d_f$ in order to compute a degree bounded solution.

4.5.1 The Algorithm

Algorithm 7: Compute $\frac{1}{g_{\min}} f$

```
Input: $d_f \ge \deg(f), d_g \ge \deg(g_{\min}),\ d_A \ge \deg(A(u))$,
    a stream $(\hat{A}^{[\ell]}, \hat{b}^{[\ell]})$  # the one that were not found to be erroneous by
    # Algorithm 6
    $R \ge \big|\{\ell \mid \hat{A}^{[\ell]} f(\xi_\ell) = g(\xi_\ell)\hat{b}^{[\ell]} \text{ and } \operatorname{rank}(\hat{A}^{[\ell]}) < r, 0 \le \ell \le L_{\text{CAB}} - 1\}\big|$,
    $E \ge \big|\{\lambda \mid \hat{A}^{[\lambda]} f(\xi_\lambda) \ne g_{\min}(\xi_\lambda)\hat{b}^{[\lambda]}, 0 \le \lambda \le L_{\text{CAB}} - 1\}\big|$
Output: $\frac{1}{g_{\min}} f$
1: $L_{\text{CAB}} \leftarrow \max\{d_A + d_f, d_b + d_g\} + R + 1$
2: Solve
                $\Psi(\xi_\ell)\hat{b}^{[\ell]} - \hat{A}^{[\ell]}\Phi(\xi_\ell) = 0$        (4.14)
    # where $\deg(\Phi) \le d_f$ and $\deg(\Psi) \le d_g$
3: $g_{\min} \leftarrow \Psi_{\min}$  # such that $\Psi_{\min} \ne 0$
4: $f \leftarrow \Phi_{\min}$  # the $\Phi$ that correspond the $\Psi_{\min}$
5: return $\frac{1}{g_{\min}} f$
```

4.6 Summary

In chapter 2 we discuss early termination strategies of an algorithm solving full rank parametric linear systems in one variable. The algorithm we consider in the subject of the [Boyer and Kaltofen 2014] Symbolic Numeric Computation paper. The algorithm works well for full rank systems even if some evaluations return erroneous results. The algorithms determines simultaneously a degree bounded solution for the parametric linear system as well as the error locations. This algorithm however, fails when applied to parametric linear systems where the matrix of the system is rank deficient and some evaluations are erroneous . In this chapter we present an algorithm for computing a degree bounded solution as well as the error locations for the rank deficient case.

Unlike the algorithm for the full rank case our algorithm does not determine the error locations at the same time that it computes the degree bounded solution. For the rank deficient algorithm we first determine the error locations. Once we locate the error(s) we can apply the full rank algorithm to the error free system in order to determine a degree bounded solution. The main idea for locating the errors is separate all errors into two mutually exclusive sets. An evaluation is in the first set of errors if the null space of the scalar matrix is not identical to the null space of the evaluated matrix. We term such errors Matrix Errors. We show that matrix errors occur at an evaluation if and only if the scalar matrix returned by the black box does not have the same null space as the evaluated matrix. We thus discover these errors by interpolating degree bounded solutions for a subspace of the null space of original matrix via the null space of the scalar matrices returned by the black box.

An evaluation point is in the second set of errors if the scalar matrix returned by the black box has the same null space as the evaluated matrix but the right side vector causes and an error. That is there exists a right side vector that would render the evaluation point error free, but the black box did not return that vector. Such errors we term Right Side Vector Errors. Recall that in the full rank case unlike the rank deficient case the solution of minimal degree in the denominator and monic is unique. Since we compute a unique solution for the full rank case then every error in the full rank case is a Right Side Vector Error. Thus the full rank algorithm works well to compute Right Side Vector Errors. So for the rank deficient algorithm our strategy is to first determine the set of linear independent columns of the scalar matrices then apply the full rank system algorithm in order to determine the Right Side Vector Errors.

There is some work to be done when some columns along with the right side vector evaluates to zero in order to ensure there are no false positives in our error reporting. We say how this is done in Step 5 to 11 of Algorithm 6. Once the errors are discovered they can be removed. Thus we can apply the full rank algorithm to an error free system in order to compute a degree bounded solution. See Algorithm 7.

Chapter 5

Polynomial Vector Recovery with Burst Errors

5.1 Introduction

So far we have considered the error locations to be random. That is, the errors we have considered are independent of each other. It is possible, however, that error locations are dependent and errors are likely to occur successively. For instance, in the polynomial vector recovery case it is possible that when an evaluation causes an error it affects the entire vector and not just a single entry in the vector. We will refer to errors that occur in succession as burst errors. Such errors patterns are the subject of this chapter and the next.

We have shown that if an erroneous evaluation may only affect one entry in the vector then for every entry in the vector we need an extra evaluation for each possible error in order to recover the vector. Hence the $2E$ in both the Generalized Welch/Berlekamp count and the Cabay count. See [Boyer and Kaltofen 2014], [Kaltofen, Pernet, Storjohann, and Waddell 2017]. So the system we solved is overdetermined. We show here that if the errors were to affect the entire vector then we only need an extra evaluation for one of the entries in the vector. The system we then solve is no longer overdetermined.

It is possible that some errors are burst errors while some are not. We show that if we have a bound for the burst errors as well as a bound for the non burst errors we can reduce the size of the system needed to recover the vector.

5.2 Reducing the size of the system

Let $f = \begin{bmatrix} f^{[1]}(u) \\ f^{[2]}(u) \\ \vdots \\ f^{[m]}(u) \end{bmatrix}$ be a black box vector of polynomials over a field K. That is, for all $i : 1 \leq i \leq m$ we have $f^{[i]} \in \mathsf{K}[u]$. Let $\gamma_i^{[\ell]} = f^{[i]}(\xi_\ell)$. We assume we can query the black box with values $\xi_\ell \in \mathsf{K}$ and the black box returns a vector of scalars $\beta^{[\ell]}$. We further assume that for k_{ERR} values $\xi_{\lambda_1}, \xi_{\lambda_2}, \ldots, \xi_{\lambda_{k_{\text{ERR}}}}$, we have that $\beta_i^{[\lambda_j]} \neq \gamma_i^{[\lambda_j]}$ for all $i : 1 \leq i \leq m$ and for all $j : 1 \leq j \leq k_{\text{ERR}}$. The evaluations ξ_{λ_j} are erroneous evaluations. Errors that affect the entire vector as previously described are burst errors.

Consider the equation

$$\gamma_i^{[\ell]} \Psi(\xi_\ell) - \beta_i^{[\ell]} \Psi(\xi_\ell) = 0, \tag{5.1}$$

where $\Psi(u)$ is a polynomial multiple of the error locator polynomial, $\Lambda(u)$. The error locator polynomial $\Lambda(u)$ is such that for all $1 \leq j \leq k_{\text{ERR}}$, $\Lambda(\xi_{\lambda_j}) = 0$.

Let $L \geq d_f + 2k_{\text{ERR}} + 1$, be the number of queries to the black box. Then for each i, we have a Reed-Solomon problem

$$\Phi^{[i]}(\xi_\ell) - \beta_i^{[\ell]} \Psi(\xi_\ell) = 0, \tag{5.2}$$

where $\deg(\Phi) = d_f + k_{\text{ERR}}$ and $\deg(\Psi) = k_{\text{ERR}}$.

Equation (5.2) is linear in the coefficients of Φ and Ψ. There are $m(d_f + k_{\text{ERR}} + 1)$ unknowns for Φ, $k_{\text{ERR}} + 1$ unknowns for Ψ, and mL equations. Consider the following matrices

$$V = \begin{bmatrix} 1 & \xi_0 & \ldots & \xi_0^{d_f + k_{\text{ERR}}} \\ 1 & \xi_1 & \ldots & \xi_1^{d_f + k_{\text{ERR}}} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \xi_{L-1} & \ldots & \xi_{L-1}^{d_f + k_{\text{ERR}}} \end{bmatrix}, \quad W_i = \begin{bmatrix} \beta_i^{[0]} & \beta_i^{[0]}\xi_0 & \ldots & \beta_i^{[0]}\xi_0^{k_{\text{ERR}}} \\ \beta_i^{[1]} & \beta_i^{[1]}\xi_1 & \ldots & \beta_i^{[1]}\xi_0^{k_{\text{ERR}}} \\ \vdots & \vdots & \ddots & \vdots \\ \beta_i^{[L-1]} & \beta_i^{[L-1]}\xi_{L-1} & \ldots & \beta_i^{[L-1]}\xi_{L-1}^{k_{\text{ERR}}} \end{bmatrix}.$$

Then the matrix

$$A = \begin{bmatrix} V & 0 & \ldots & 0 & -W_1 \\ 0 & V & \ldots & 0 & -W_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \ldots & V & -W_m \end{bmatrix}$$

is the coefficient matrix for equation (5.2). We select a sub-matrix, denoted by A^* from A by removing $k_{\text{ERR}}(m-1)$ many rows from all but one Reed-Solomon problem. For instance we form the sub-matrix A^* by removing k_{ERR} rows from all but the rows corresponding to the first block of A.

Lemma 8 *Let $V \in \mathsf{K}^{m\times n}$ be a Vandermonde matrix, where $m > n$, that is $V_{i,j} = \xi_i^{j-1}$.*
Let $W_1 \in \mathsf{K}^{m\times p}$ and $W_{1_{i,j}} = f(\xi_i)\xi_i^{j-1}$.
Let $f \in \mathsf{K}[u]$, such that $\deg(f) = k \leq n-p$.
Then $\operatorname{rank}(V \mid W_1) = \operatorname{rank}(V)$.

Proof. Any square Vandermonde matrix is invertible. This implies that any of the last $m-n$ rows of V can be written as a linear combination of the first n rows of V. We will show that any of the last $m-n$ rows of $(V \mid W_1)$ is a linear combinations of its first n rows. In fact, the same combination that works for a row $k > n$ in V also works for the corresponding row in $(V \mid W_1)$. It is enough to show that if $V_{k,*} = \sum_{i=1}^n c_i V_{i,*}$ then $W_{1_{k,*}} = \sum_{i=1}^n c_i W_{1_{i,*}}$.

$$\begin{aligned}
V_{k,*} &= \sum_{i=1}^{n} c_i V_{i,*} \\
&= c_1 \begin{bmatrix} \xi_1^0 & \xi_1^1 & \ldots & \xi_1^{n-1} \end{bmatrix} + c_2 \begin{bmatrix} \xi_2^0 & \xi_2^1 & \ldots & \xi_2^{n-1} \end{bmatrix} + \ldots + c_n \begin{bmatrix} \xi_n^0 & \xi_n^1 & \ldots \xi_n^{n-1} \end{bmatrix} \\
&= \begin{bmatrix} \xi_k^0 & \xi_k^1 & \ldots & \xi_k^{n-1} \end{bmatrix}.
\end{aligned}$$

Observe that $f(u) = a_0 + a_1 u + \ldots + a_k u^k$.

$$\begin{aligned}
\sum_{i=1}^{n} c_i W_{1_{i,*}} &= c_1 f(\xi_1) \begin{bmatrix} \xi_1^0 & \xi_1^1 & \ldots & \xi_1^{p-1} \end{bmatrix} + c_2 f(\xi_2) \begin{bmatrix} \xi_2^0 & \xi_2^1 & \ldots & \xi_2^{p-1} \end{bmatrix} + \ldots + c_n f(\xi_n) \begin{bmatrix} \xi_n^0 & \xi_n^1 & \ldots & \xi_n^{p-1} \end{bmatrix} \\
&= c_1(a_0 + a_1\xi_1 + \ldots + a_k\xi_1^k) \begin{bmatrix} \xi_1^0 & \xi_1^1 & \ldots & \xi_1^{p-1} \end{bmatrix} + c_2(a_0 + a_1\xi_2 + \ldots + a_k\xi_2^k) \begin{bmatrix} \xi_2^0 & \xi_2^1 & \ldots & \xi_2^{p-1} \end{bmatrix} \\
&+ \ldots + c_n(a_0 + a_1\xi_n + \ldots + a_k\xi_n^k) \begin{bmatrix} \xi_n^0 & \xi_n^1 & \ldots & \xi_n^{p-1} \end{bmatrix} \\
&= c_1 \begin{bmatrix} \sum\limits_{j=0}^{k} a_j \xi_1^j & \sum\limits_{j=0}^{k} a_j \xi_1^{j+1} & \ldots & \sum\limits_{j=0}^{k} \xi_1^{j+p-1} \end{bmatrix} + c_2 \begin{bmatrix} \sum\limits_{j=0}^{k} a_j \xi_2^j & \sum\limits_{j=0}^{k} a_j \xi_2^{j+1} & \ldots & \sum\limits_{j=0}^{k} \xi_2^{j+p-1} \end{bmatrix} \\
&+ \ldots + c_n \begin{bmatrix} \sum\limits_{j=0}^{k} a_j \xi_n^j & \sum\limits_{j=0}^{k} a_j \xi_n^{j+1} & \ldots & \sum\limits_{j=0}^{k} \xi_n^{j+p-1} \end{bmatrix} \\
&= \begin{bmatrix} \sum\limits_{j=0}^{k} a_j \left(\sum\limits_{i=1}^{n} c_i \xi_i^j \right) & \sum\limits_{j=0}^{k} a_j \left(\sum\limits_{i=1}^{n} c_i \xi_i^{j+1} \right) & \ldots & \sum\limits_{j=0}^{k} a_j \left(\sum\limits_{i=1}^{n} c_i \xi_i^{j+p-1} \right) \end{bmatrix} \\
&= \begin{bmatrix} \sum\limits_{j=0}^{k} a_j \xi_k^j & \sum\limits_{j=0}^{k} a_j \xi_k^{j+1} & \ldots & \sum\limits_{j=0}^{k} a_j \xi_k^{j+p-1} \end{bmatrix} \\
&= \begin{bmatrix} f(\xi_k) & \xi_k f(\xi_k) & \ldots & \xi_k^{p-1} f(\xi_k) \end{bmatrix} \\
&= f(\xi_k) \begin{bmatrix} \xi_k^0 & \xi_k & \ldots & \xi_k^{p-1} \end{bmatrix} \\
&= W_{1_{k,*}}.
\end{aligned}$$

□

Lemma 9 *Let $V \in \mathsf{K}^{n+p \times n}$ be such that $V_{i,j} = \xi_{i-1}^{j-i}$ and let $W_1 \in \mathsf{K}^{n+p \times p}$ be such that $W_{1_{i,j}} = \beta_i \xi_{i-1}^{j-i}$, where the ξ_i's are distinct. Let $f \in \mathsf{K}[u]$ such that* $\deg(f) = d \leq n - p$, *where p is such that $1 \leq p \leq n-1$. Assume that for p many ξ_λ's we have that $\beta_\lambda \neq f(\xi_\lambda)$. Let Y be an $n \times n$ submatrix of V where the entries are such that $\beta_i = f(\xi_i)$. Let Z be the $p \times p$ submatrix of W_1 where the corresponding entries in W_1 are such that $\beta_\lambda \neq f(\xi_\lambda)$. Then the determinant of $(V \mid \beta W_1)$ is*

$$(-1)^p \det(Y) \det(Z) \prod_{i=1}^{p} (\beta_{\lambda_i} - f(\xi_{\lambda_i})).$$

Proof. Let $Y \in \mathsf{K}^{n\times n}$ and $X \in \mathsf{K}^{n\times p}$ such that

$$Y = \begin{bmatrix} 1 & \xi_0 & \xi_0^2 & \dots & \xi_0^{n-1} \\ 1 & \xi_1 & \xi_1^2 & \dots & \xi_1^{n-1} \\ 1 & \xi_2 & \xi_0^2 & \dots & \xi_2^{n-1} \\ \vdots & \vdots & \dots & \vdots & \vdots \\ 1 & \xi_{n-1} & \xi_{n-1}^2 & \dots & \xi_{n-1}^{n-1} \end{bmatrix} \quad \text{and}$$

$$X = \begin{bmatrix} f_0 + f_1\xi_0 + f_2\xi_0^2 + \ldots + f_d\xi_0^d & f_0\xi_0 + f_1\xi_0^2 + f_2\xi_0^3 + \ldots + f_d\xi_0^{d+1} & \dots & f_0\xi_0^{p-1} + f_1\xi_0^p + f_2\xi_0^{p+1} + \ldots + f_d\xi_0^{d+p-1} \\ f_0 + f_1\xi_1 + f_2\xi_1^2 + \ldots + f_d\xi_1^d & f_0\xi_1 + f_1\xi_1^2 + f_2\xi_1^3 + \ldots + f_d\xi_1^{d+1} & \dots & f_0\xi_1^{p-1} + f_1\xi_1^p + f_2\xi_1^{p+1} + \ldots + f_d\xi_1^{2p-1} \\ f_0 + f_1\xi_2 + f_2\xi_2^2 + \ldots + f_d\xi_2^d & f_0\xi_2 + f_1\xi_2^2 + f_2\xi_2^3 + \ldots + f_d\xi_2^{d+1} & \dots & f_0\xi_2^{p-1} + f_1\xi_2^p + f_2\xi_2^{p+1} + \ldots + f_d\xi_2^{d+p-1} \\ \vdots & \vdots & \dots & \vdots \\ f_0 + f_1\xi_{n-1} + f_2\xi_{n-1}^2 + \ldots + f_d\xi_{n-1}^d & f_0\xi_{n-1} + f_1\xi_{n-1}^2 + f_2\xi_{n-1}^3 + \ldots + f_d\xi_{n-1}^{d+1} & \dots & f_0\xi_{n-1}^{p-1} + f_1\xi_{n-1}^p + f_2\xi_{n-1}^{p+1} + \ldots + f_{p-1}\xi_{n-1}^{d+p-1} \end{bmatrix}$$

Let $U \in \mathsf{K}^{p\times n}$ and $Z \in \mathsf{K}^{p\times p}$ such that

$$U = \begin{bmatrix} 1 & \xi_n & \xi_n^2 & \dots & \xi_n^{n-1} \\ 1 & \xi_{n+1} & \xi_{n+1}^2 & \dots & \xi_{n+1}^{n-1} \\ 1 & \xi_{n+2} & \xi_{n+2}^2 & \dots & \xi_{n+2}^{n-1} \\ \vdots & \vdots & \vdots & \dots & \vdots \\ 1 & \xi_{n+p-1} & \xi_{n+p-1}^2 & \dots & \xi_{n+p-1}^{n-1} \end{bmatrix} \quad \text{and} \quad Z = \begin{bmatrix} 1 & \xi_{\lambda_1} & \xi_{\lambda_1}^2 & \dots & \xi_{\lambda_1}^{n-1} \\ 1 & \xi_{\lambda_2} & \xi_{\lambda_2}^2 & \dots & \xi_{\lambda_2}^{n-1} \\ 1 & \xi_{\lambda_3} & \xi_{\lambda_3}^2 & \dots & \xi_{\lambda_3}^{n-1} \\ \vdots & \vdots & \vdots & \dots & \vdots \\ 1 & \xi_{\lambda_p} & \xi_{\lambda_p}^2 & \dots & \xi_{\lambda_p}^{n-1} \end{bmatrix}$$

Observe that

$$(V \mid \beta W) = \begin{bmatrix} Y & X \\ U & \beta Z \end{bmatrix}$$

So the $\det((V \mid \beta W)) = \det(Y)\det(\beta Z - UY^{-1}X)$. Let $S \in \mathsf{K}^{n\times d+p}$ and $T \in \mathsf{K}^{d+p\times p}$ such that

$$S = \begin{bmatrix} 1 & \xi_0 & \xi_0^2 & \dots & \xi_0^{d+p-1} \\ 1 & \xi_1 & \xi_1^2 & \dots & \xi_1^{d+p-1} \\ 1 & \xi_2 & \xi_0^2 & \dots & \xi_2^{d+p-1} \\ \vdots & \vdots & \dots & \vdots & \vdots \\ 1 & \xi_{n-1} & \xi_{n-1}^2 & \dots & \xi_{n-1}^{d+p-1} \end{bmatrix} \quad \text{and} \quad T = \begin{bmatrix} f_0 & 0 & 0 & \dots & 0 \\ f_1 & f_0 & 0 & \dots & 0 \\ f_2 & f_1 & f_0 & \dots & 0 \\ \vdots & \vdots & \vdots & \dots & 0 \\ f_{d-1} & f_{d-2} & f_{d-3} & \dots & 0 \\ f_d & f_{d-1} & f_{d-2} & \dots & 0 \\ 0 & f_d & f_{d-1} & \dots & 0 \\ \vdots & \vdots & \vdots & \dots & \vdots \\ 0 & 0 & 0 & \dots & f_d \end{bmatrix}$$

Observe that $X = ST$. So the $\det((V \mid \beta W_1)) = \det(Y)\det(\beta Z - UY^{-1}ST)$.
Since $d + p \leq n$ and matrix multiplication is associative we have that

$$Y^{-1}S = \begin{bmatrix} I \\ 0 \end{bmatrix} \in \mathsf{K}^{n\times d+p}$$

where $I \in \mathsf{K}^{d+p\times d+p}$. Then

$$Y^{-1}ST = \begin{bmatrix} T \\ 0 \end{bmatrix} \in \mathsf{K}^{n\times p}$$

Then

$$UY^{-1}ST = \begin{bmatrix} f(\xi_{\lambda_1}) & f(\xi_{\lambda_1})\xi_{\lambda_1} & f(\xi_{\lambda_1})\xi_{\lambda_1}^2 & \dots & f(\xi_{\lambda_1})\xi_{\lambda_1}^{n-1} \\ f(\xi_{\lambda_2}) & f(\xi_{\lambda_2})\xi_{\lambda_2} & f(\xi_{\lambda_2})\xi_{\lambda_2}^2 & \dots & f(\xi_{\lambda_2})\xi_{\lambda_2}^{n-1} \\ f(\xi_{\lambda_3}) & f(\xi_{\lambda_3})\xi_{\lambda_3} & f(\xi_{\lambda_3})\xi_{\lambda_3}^2 & \dots & f(\xi_{\lambda_3})\xi_{\lambda_3}^{n-1} \\ \vdots & \vdots & \vdots & \dots & \vdots \\ f(\xi_{\lambda_p}) & f(\xi_{\lambda_p})\xi_{\lambda_p} & f(\xi_{\lambda_p})\xi_{\lambda_p}^2 & \dots & f(\xi_{\lambda_p})\xi_{\lambda_p}^{n-1} \end{bmatrix} \in \mathsf{K}^{p\times p}$$

and

$$\beta Z - UY^{-1}ST = \begin{bmatrix} \beta_{\lambda_1} - f(\xi_{\lambda_1}) & (\beta_{\lambda_1} - f(\xi_{\lambda_1}))\xi_{\lambda_1} & (\beta_{\lambda_1} - f(\xi_{\lambda_1}))\xi_{\lambda_1}^2 & \dots & (\beta_{\lambda_1} - f(\xi_{\lambda_1}))\xi_{\lambda_1}^{n-1} \\ \beta_{\lambda_2} - f(\xi_{\lambda_2}) & (\beta_{\lambda_2} - f(\xi_{\lambda_2}))\xi_{\lambda_2} & (\beta_{\lambda_2} - f(\xi_{\lambda_2}))\xi_{\lambda_2}^2 & \dots & (\beta_{\lambda_2} - f(\xi_{\lambda_2}))\xi_{\lambda_2}^{n-1} \\ \beta_{\lambda_3} - f(\xi_{\lambda_3}) & (\beta_{\lambda_3} - f(\xi_{\lambda_3}))\xi_{\lambda_3} & (\beta_{\lambda_3} - f(\xi_{\lambda_3}))\xi_{\lambda_3}^2 & \dots & (\beta_{\lambda_3} - f(\xi_{\lambda_3}))\xi_{\lambda_3}^{n-1} \\ \vdots & \vdots & \vdots & \dots & \vdots \\ \beta_{\lambda_p} - f(\xi_{\lambda_p}) & (\beta_{\lambda_p} - f(\xi_{\lambda_p}))\xi_{\lambda_p} & (\beta_{\lambda_p} - f(\xi_{\lambda_p}))\xi_{\lambda_p}^2 & \dots & (\beta_{\lambda_p} - f(\xi_{\lambda_p}))\xi_{\lambda_p}^{n-1} \end{bmatrix} \in \mathsf{K}^{p\times p}$$

Thus

$$\det(\beta Z - UY^{-1}X) = (-1)^p \det(Z) \prod_{i=1}^{p} (\beta_{\lambda_i} - f(\xi_{\lambda_i})).$$

Therefore,

$$\det(V \mid \beta W_1) = (-1)^p \det(Y) \det(Z) \prod_{i=1}^{p} (\beta_{\lambda_i} - f(\xi_{\lambda_i})).$$

□

Corollary 2 *The solution of a Reed-Solomon system where $k_{\text{ERR}} = E$ is unique.*

Lemma 10 *If for all $i : 1 \le i \le m$ the ordered pair $(\Phi^{[i]}(u), \Psi(u))$ is a solution to equation (5.2) then we have that $\Lambda \mid \Phi^{[i]}$.*

Proof. Observe that the first L rows of the remaining system, A^*, is a complete Reed-Solomon system with exactly the $\deg(\Psi)$ many errors. Thus, by corollary 2, the system in the first block has a unique solution for $\Phi^{[1]}$ and Ψ. So for our reduced system, with matrix A^*, the coefficients of $\Phi^{[1]}$ and Ψ are determined and we have $f^{[1]} = \Phi^{[1]}/\Psi$. In fact, $\Psi(u) = \Lambda(u) = \prod_{j=1}^{k_{\text{ERR}}}(x - \xi_{\lambda_j})$, see [Boyer and Kaltofen 2014; Kaltofen, Pernet, Storjohann, and Waddell 2017]. Next we show that for all $i : 1 \le i \le m$ we have that $\Lambda \mid \Phi^{[i]}$. We already have this for $i = 1$. For all $i : 1 < i \le m$, we have $\Phi^{[i]}(\xi_\ell) - \beta_i^{[\ell]}\Psi(\xi_\ell) = 0$, so $\Phi^{[i]}(\xi_\ell) = \beta_i^{[\ell]}\Psi(\xi_\ell)$. Since $\Psi(u) = \Lambda(u)$ we have that $\Phi^{[i]}(\xi_\ell) = \beta_i^{[\ell]}\Lambda(\xi_\ell)$. Now for all $i : 1 < i \le m$ we have that $\ge d_f + E + 1$ evaluations are such that $\Phi^{[i]}(\xi_\ell) = \gamma_i^{[\ell]}\Lambda(\xi_\ell)$, so $\Phi^{[i]}(\xi_\ell) = f^{[i]}(\xi_\ell)\Lambda(\xi_\ell)$. Since for all $i : 1 < i \le m$ we have $\deg(f^{[i]}(u)\Lambda(u)) \le d_f + E$ and $\deg(\Phi^{[i]}) \le d_f + E$ it must be that $\Phi^{[i]}(u) = f^{[i]}(u)\Lambda(u)$. Hence $\Lambda(u) \mid \Phi^{[i]}(u)$ for all $i : 1 \le i \le m$. □

Remark 13 For our proof we do not need an erroneous evaluation, ξ_{λ_j}, to affect the entire vector $\beta^{[\lambda_j]}$. We only need that for all erroneous evaluations, ξ_{λ_j}, we have $\beta_1^{[\lambda_j]} \ne \gamma_1^{[\lambda_j]}$. In other words if there is an evaluation that causes the i^{th} entry of $\beta^{[\lambda]}$ to be wrong, then the 1^{st} entry of $\beta^{[\lambda]}$ is also wrong. One way to enforce this is to interleave the polynomials we wish to recover rather than trying to recover them individually. We save evaluations because we only need an extra good evaluation for each erroneous evaluation in the first block. Thus k_{ERR} evaluations can be removed from all other Reed-Solomon problems. This is possible because, as we demonstrate in the proof of lemma 10, the error locator polynomial is uniquely determined by the first block and shared with all other blocks. □

Now let's assume we do not know the number of burst errors, k_{ERR}, that occurred. We assume, however, that we have an upper bound, E_{BRST}, for the number of burst errors. Let $L \ge d_f + 2E_{\text{BRST}} + 1$, be the number of evaluations. Then for each i we have a Reed-Solomon problem

$$\Phi^{[i]}(\xi_\ell) - \beta_i^{[\ell]}\Psi(\xi_\ell) = 0, \tag{5.3}$$

where $\deg(\Phi) = d_f + E_{\text{BRST}}$ and $\deg(\Psi) = E_{\text{BRST}}$.

Equation (5.3) is linear in the coefficient of Φ and Ψ. There are $m(d_f + E_{\text{BRST}} + 1)$ unknowns for Φ, $E_{\text{BRST}} + 1$ unknowns for Ψ, and mL equations.

Consider the following matrices

$$M = \begin{bmatrix} 1 & \xi_0 & \dots & \xi_0^{d_f+E_{\text{BRST}}} \\ 1 & \xi_1 & \dots & \xi_1^{d_f+E_{\text{BRST}}} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \xi_{L-1} & \dots & \xi_{L-1}^{d_f+E_{\text{BRST}}} \end{bmatrix}, \quad W_i = \begin{bmatrix} \beta_i^{[0]} & \beta_i^{[0]}\xi_0 & \dots & \beta_i^{[0]}\xi_0^{E_{\text{BRST}}} \\ \beta_i^{[1]} & \beta_i^{[1]}\xi_1 & \dots & \beta_i^{[1]}\xi_1^{E_{\text{BRST}}} \\ \vdots & \vdots & \ddots & \vdots \\ \beta_i^{[L-1]} & \beta_i^{[L-1]}\xi_{L-1} & \dots & \beta_i^{[L-1]}\xi_{L-1}^{E_{\text{BRST}}} \end{bmatrix}.$$

Then the matrix

$$A = \begin{bmatrix} M & 0 & \dots & 0 & -W_1 \\ 0 & M & \dots & 0 & -W_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & M & -W_m \end{bmatrix}$$

is the coefficient matrix for equation (5.3). We create a sub-matrix of A^* of A by removing the last $E_{\text{BRST}}(m-1)$ rows from all but the first block A.

Lemma 11 *If for all $i : 1 \le i \le m$ the ordered pair $(\Phi^{[i]}(u), \Psi(u))$ is a solution to equation (5.3) then the ordered pair $(\Phi_{\min}, \Psi_{\min}) = (f\Lambda, \Lambda)$. The polynomial $\Psi_{\min}$ has minimum degree among the denominators of all solutions to (5.3) and $\Phi_{\min}$ is the corresponding numerator.*

Proof. We again have a complete Reed - Solomon system in the first block, however $\deg(\Psi) \ge \deg(\Lambda)$. By construction the minimum degree that Ψ can be is k_{ERR}, the degree of Λ. We know that if we have two or more solutions for (5.3) then any linear combination of those solutions will produce another solution. Thus the is only one monic polynomial Ψ of degree k_{ERR} We know from Theorem 10 that there exists a square submatrix of $A^*_{E_{\text{BRST}}}$ for which $(f\Lambda, \Lambda)$ is the unique solution. Observe that this solution accounts for all erroneous evaluations in equation (5.3) and that for all other evaluations $(f\Lambda, \Lambda)$ is a solution to equation (5.3). Thus $(f\Lambda, \Lambda)$ solves equation (5.3). Since this solution is unique among all other solutions of equation (5.3) we have that $(\Phi_{\min}, \Psi_{\min}) = (f\Lambda, \Lambda)$. □

Corollary 3 *We can always reduce the size of the system we need to solve if we have a bound for the number of burst errors.*

Remark 14 We know that we can recover a rational function vector from $L = d_f + d_g + 2E + 1$ many evaluations by solving the linear system

$$\Phi^{[i]}(\xi_\ell) - \beta_i^{[\ell]}\Psi(\xi_\ell) = 0, \tag{5.4}$$

where for all $i : \deg(\Phi^{[i]}) = d_f + E$ and $\deg(\Psi) = d_g + E$. See [Kaltofen, Pernet, Storjohann, and Waddell 2017]. Thus we can recover a polynomial vector from $L = d_f + 2E + 1$ many evaluations and solve equation (5.4) since for a polynomial vector $\deg(g) = 0$. Suppose we know that some errors are burst errors and we can divide our error bound into two types of errors E_{BRST} and E_{NBRST} such that $E_{\text{BRST}} + E_{\text{NBRST}} = E$. Then we can remove E_{BRST} many rows from all but one block in equation (5.4) before solving for the coefficients of Φ and Ψ. Thus we recover the polynomial vector from a solution of a smaller system than that used in [Boyer and Kaltofen 2014] and [Kaltofen, Pernet, Storjohann, and Waddell 2017], where burst errors are not considered. The proof follows from lemmas 10 and 11 and Theorem 2.2 in [Boyer and Kaltofen 2014]. $\square$

5.3 Summary

For errors that affect every entry in the vector we only need an extra true evaluation for one of the entries in the vector to ensure we always recover the entries in the vector. It does not matter which entry we choose as long as it is the same entry for each possible error. It should be noted that we not need to first find the error locations before we can recover the vector. Our algorithm simultaneously recovers the vectors and locates the errors.

For errors that do not affect the entire vector we need a extra good evaluation for every entry in the vector for all such errors in order to ensure that we can recover the vector.

Chapter 6

Modeling Polynomial Vector Recovery as a Burst Error Correcting Code

6.1 Introduction

In the previous chapter we have shown that when errors occur in bursts we can reduce the size of the linear system necessary to recover the polynomials as well as the error locator polynomial. This implies that we can recover the polynomials from fewer evaluations. The problem we present in this chapter, though related to that of the previous chapter, is different. In this setting we think of the coefficients of the polynomials we wish to recover as components of a message that was sent over a noisy network that produces burst errors. In this setting polynomial recovery is a type of Reed-Solomon Error Correcting Code and polynomial vector recovery can be thought of as interleaved Reed-Solomon Codes. It is known that interleaved codes are useful for decoding on channels that produce burst errors. In fact,[Leung and Welch 1981; Yee and Weldon 1995; Wolf 1998], interleaved Reed-Solomon codes are among the best known burst error correcting codes. The problem we solve in this chapter is as follows: given a message and information about the burst errors decide on an interleaving scheme. The interleaving scheme must take advantage of the fact that the channel produces burst errors and allow for fewer data to be sent than would be required if each polynomial was recovered individually or there were no burst errors. We know that when errors occur in bursts we are able to use fewer evaluation to recover the polynomial vector. This means that we can send fewer data across the network than we would be able to if they were no burst errors or if we were recovering the polynomials individually. The idea is to reduce the redundancy in the codeword.

Recall that the system we solve in order to recover a polynomial vector is over determined.

That is there are more rows than there are columns. In order to reduce the size of the system we remove E_{BRST} many rows from all but one polynomial. Recall that each row constitutes one evaluation of a polynomial in the vector. The idea is that, in the error correcting code setting, we would like to transmit only the evaluations that remain after E_{BRST} many evaluations have been removed from all but one of the polynomials in polynomial vector recovery setting. See tables 6.1, 6.2, and 6.3. This means that we always send more data that pertains to one of the polynomials. Since the error occurs in bursts we cannot send the extra data for that polynomial sequentially because if there are too many errors in that polynomial then we will not be able to recover the vector. We must then design our interleave, table 6.4, so that errors to do accumulate in one particular polynomial. In order to enforce this we require as much distance as possible between transmissions of the same polynomial and a constant interleave depth. This ideas are illustrated in the tables of Section 6.2.

6.2 Problem Description

Definition 7 ***Error Correcting Code:*** *An error correcting code is a process of adding redundant or parity data to a message so that it can be recovered by a receiver even if errors are induced during transmission.*

Definition 8 ***Codeword:*** *A codeword is the message plus the redundant or parity data added to the message in order to aid in recovery of the message by the receiver.*

Definition 9 ***Received word:*** *The received word is what the receiver gets.*

Definition 10 ***Errors:*** *An error is any place where the codeword differs from the received word.*

Definition 11 ***Burst Error Model:*** *A scheme in which errors are likely to occur successively.*

Consider:

1. the coefficients of the polynomials to be the message,

2. the true evaluations of the polynomials to be the codewords,

3. the evaluations returned by the black box to be the received word

then each polynomial in the polynomial vector recovery problem can be viewed as a Reed-Solomon code. Therefore polynomial vector recovery is essentially an interleaved Reed-Solomon code.

Example 4 *For ease of explanation assume that only burst errors occur. Later we will discuss situations where we have a mix of burst and non-burst errors. Further assume that given a message and an upper bound for the number of burst errors, say $E_{\text{BRST}} = 4$, we computed the degree of the polynomials needed for our interleave scheme to be $\deg(f) = 3$. Then the generalized Welch/Berlekamp bound $L_{\text{BK}} = \deg(f) + 2E_{\text{BRST}} + 1 = 12$. Recall that $\gamma_i^{[\ell]} = f^{[i]}(\xi_\ell)$.*

Table 6.1: Evaluated vectors before $(m-1)E_{\text{BRST}}$ many evaluations are removed

$\gamma_1^{[0]}$	$\gamma_1^{[1]}$	$\gamma_1^{[2]}$	$\gamma_1^{[3]}$	$\gamma_1^{[4]}$	$\gamma_1^{[5]}$	$\gamma_1^{[6]}$	$\gamma_1^{[7]}$	$\gamma_1^{[8]}$	$\gamma_1^{[9]}$	$\gamma_1^{[10]}$	$\gamma_1^{[11]}$
$\gamma_2^{[0]}$	$\gamma_2^{[1]}$	$\gamma_2^{[2]}$	$\gamma_2^{[3]}$	$\gamma_2^{[4]}$	$\gamma_2^{[5]}$	$\gamma_2^{[6]}$	$\gamma_2^{[7]}$	$\gamma_2^{[8]}$	$\gamma_2^{[9]}$	$\gamma_2^{[10]}$	$\gamma_2^{[11]}$
$\gamma_3^{[0]}$	$\gamma_3^{[1]}$	$\gamma_3^{[2]}$	$\gamma_3^{[3]}$	$\gamma_3^{[4]}$	$\gamma_3^{[5]}$	$\gamma_3^{[6]}$	$\gamma_3^{[7]}$	$\gamma_3^{[8]}$	$\gamma_3^{[9]}$	$\gamma_3^{[10]}$	$\gamma_3^{[11]}$
$\gamma_4^{[0]}$	$\gamma_4^{[1]}$	$\gamma_4^{[2]}$	$\gamma_4^{[3]}$	$\gamma_4^{[4]}$	$\gamma_4^{[5]}$	$\gamma_4^{[6]}$	$\gamma_4^{[7]}$	$\gamma_4^{[8]}$	$\gamma_4^{[9]}$	$\gamma_4^{[10]}$	$\gamma_4^{[11]}$

Table 6.1 refers to example 4. It shows the outputs of the polynomials at L_{BK} many evaluations. Each row is a Reed-Solomon codeword that corresponds to a polynomial in the vector.

Table 6.2: Evaluated vectors after $(m-1)E_{\text{BRST}}$ many evaluations are removed

$\gamma_1^{[0]}$	$\gamma_1^{[1]}$	$\gamma_1^{[2]}$	$\gamma_1^{[3]}$	$\gamma_1^{[4]}$	$\gamma_1^{[5]}$	$\gamma_1^{[6]}$	$\gamma_1^{[7]}$	$\gamma_1^{[8]}$	$\gamma_1^{[9]}$	$\gamma_1^{[10]}$	$\gamma_1^{[11]}$
$\gamma_2^{[0]}$	$\gamma_2^{[1]}$	$\gamma_2^{[2]}$	$\gamma_2^{[3]}$	$\gamma_2^{[4]}$	$\gamma_2^{[5]}$	$\gamma_2^{[6]}$	$\gamma_2^{[7]}$				
$\gamma_3^{[0]}$	$\gamma_3^{[1]}$	$\gamma_3^{[2]}$	$\gamma_3^{[3]}$	$\gamma_3^{[4]}$	$\gamma_3^{[5]}$	$\gamma_3^{[6]}$	$\gamma_3^{[7]}$				
$\gamma_4^{[0]}$	$\gamma_4^{[1]}$	$\gamma_4^{[2]}$	$\gamma_4^{[3]}$	$\gamma_4^{[4]}$	$\gamma_4^{[5]}$	$\gamma_4^{[6]}$	$\gamma_4^{[7]}$				

In example 4 there are only burst errors. We assume that a burst error affects every entry in the vector. There are no more than $E_{\text{BRST}} = 4$ many such errors. Thus we remove E_{BRST} many evaluations from all but one polynomial in the vector. In this case we remove 4 evaluations from all but the first polynomial. This is shown in table 6.2.

Table 6.3: Evaluated vectors to be transmitted

$\gamma_1^{[0]}$	$\gamma_1^{[1]}$	$\gamma_1^{[2]}$	$\gamma_1^{[3]}$	$\gamma_1^{[4]}$	$\gamma_1^{[5]}$	$\gamma_1^{[6]}$	$\gamma_1^{[7]}$	$\gamma_1^{[8]}$
$\gamma_2^{[0]}$	$\gamma_2^{[1]}$	$\gamma_2^{[2]}$	$\gamma_2^{[3]}$	$\gamma_2^{[4]}$	$\gamma_2^{[5]}$	$\gamma_2^{[6]}$	$\gamma_2^{[7]}$	$\gamma_1^{[9]}$
$\gamma_3^{[0]}$	$\gamma_3^{[1]}$	$\gamma_3^{[2]}$	$\gamma_3^{[3]}$	$\gamma_3^{[4]}$	$\gamma_3^{[5]}$	$\gamma_3^{[6]}$	$\gamma_3^{[7]}$	$\gamma_1^{[10]}$
$\gamma_4^{[0]}$	$\gamma_4^{[1]}$	$\gamma_4^{[2]}$	$\gamma_4^{[3]}$	$\gamma_4^{[4]}$	$\gamma_4^{[5]}$	$\gamma_4^{[6]}$	$\gamma_4^{[7]}$	$\gamma_1^{[11]}$

We have removed evaluations not necessary to recover the polynomial vector. This is done before transmission, so there are no errors. Note that the order in which the data is transmitted is always from the top of the vector to the bottom. For instance in example 4 the evaluations are transmitted in the following order:

$$\gamma_1^{[0]}, \gamma_2^{[0]}, \ldots, \gamma_4^{[0]}, \gamma_1^{[1]}, \ldots, \gamma_4^{[1]}, \ldots, \gamma_1^{[9]}, \gamma_1^{[10]}, \gamma_1^{[11]}$$

Observe that at the tail of the transmission, evaluations of the same polynomial are transmitted consecutively. If burst errors occur before and during the tail of the transmission then there maybe 5 or more errors in the first polynomial. This violates the assumptions of lemma 11 and we may not be able to recover the vector.

Table 6.4 shows an interleave scheme that contains the same information as table 6.3, but does not repeat any polynomial in the same column. Observe that the separation between

Table 6.4: Evaluated vectors interleaved for transmission

$\gamma_1^{[0]}$	$\gamma_1^{[1]}$	$\gamma_1^{[2]}$	$\gamma_1^{[3]}$	$\gamma_1^{[4]}$	$\gamma_1^{[5]}$	$\gamma_1^{[6]}$	$\gamma_1^{[7]}$	$\gamma_1^{[8]}$	$\gamma_1^{[9]}$	$\gamma_1^{[10]}$	$\gamma_1^{[11]}$
$\gamma_2^{[0]}$	$\gamma_2^{[1]}$	$\gamma_2^{[2]}$	$\gamma_2^{[3]}$	$\gamma_2^{[4]}$	$\gamma_2^{[5]}$	$\gamma_2^{[6]}$	$\gamma_2^{[7]}$	$\gamma_3^{[8]}$	$\gamma_3^{[9]}$	$\gamma_3^{[10]}$	$\gamma_3^{[11]}$
$\gamma_3^{[0]}$	$\gamma_3^{[1]}$	$\gamma_3^{[2]}$	$\gamma_3^{[3]}$	$\gamma_4^{[4]}$	$\gamma_4^{[5]}$	$\gamma_4^{[6]}$	$\gamma_4^{[7]}$	$\gamma_4^{[8]}$	$\gamma_4^{[9]}$	$\gamma_4^{[10]}$	$\gamma_4^{[11]}$

evaluations of the same polynomial is always constant.

6.3 The Interleaving Scheme

In the interleaving scheme there is always one complete Reed-Solomon codeword. This accounts for $\deg(f)+2E_{\text{BRST}}+1$ many evaluations. Since we remove E_{BRST} many evaluations from the other codewords, each will have $\deg(f)+E_{\text{BRST}}+1$ many evaluations remaining. In order to ensure that the separation between transmission of the same Reed-Solomon codeword be constant we must have that the total number of transmissions is a multiple of $\deg(f)+2E_{\text{BRST}}+1$. The interleaving scheme suggests the following equation.

$$\deg(f)+2E_{\text{BRST}}+1+c(\deg(f)+E_{\text{BRST}}+1)=\hat{c}(\deg(f)+2E_{\text{BRST}}+1) \tag{6.1}$$

Theorem 11 *If $E_{\text{BRST}}>0$ then $c\geq 3$*

Proof. First observe that $c\in\mathbb{Z}_{>0}$. We first show that $c>1$. Assume $c=1$ this implies that

$$\begin{aligned}\deg(f)+2E_{\text{BRST}}+1+\deg(f)+E_{\text{BRST}}+1 &= \hat{c}(\deg(f)+2E_{\text{BRST}}+1) \implies \hat{c}\neq 0\\ \deg(f)+E_{\text{BRST}}+1 &= (\hat{c}-1)(\deg(f)+2E_{\text{BRST}}+1) \implies \hat{c}\neq 1\end{aligned}$$

this implies $\hat{c}\in\mathbb{Z}_{\geq 2}$. This implies however, $\deg(f)+E_{\text{BRST}}+1\geq\deg(f)+2E_{\text{BRST}}+1$. This is a contradiction since $E_{\text{BRST}}>0$. Thus $c>1$.

Next we show that $c>2$. Observe that

$$\frac{\deg(f)+2E_{\text{BRST}}+1}{2}<\deg(f)+E_{\text{BRST}}+1<\deg(f)+2E_{\text{BRST}}+1 \tag{6.2}$$

Assume that $c=2$ this implies that

$$\begin{aligned}\deg(f)+2E_{\text{BRST}}+1+2(\deg(f)+E_{\text{BRST}}+1) &= \hat{c}(\deg(f)+2E_{\text{BRST}}+1)\\ 2(\deg(f)+E_{\text{BRST}}+1) &= \hat{c}(\deg(f)+2E_{\text{BRST}}+1)\\ \deg(f)+E_{\text{BRST}}+1 &= (\hat{c}-1)\frac{\deg(f)+2E_{\text{BRST}}+1}{2}\end{aligned}$$

Recall $\hat{c}\in\mathbb{Z}_{\geq 2}$. If $\hat{c}=2$ then $\deg(f)+E_{\text{BRST}}+1=\deg(f)+2E_{\text{BRST}}+1$ which is a contradiction. If $\hat{c}>2$ then $\deg(f)+E_{\text{BRST}}+1\geq\deg(f)+2E_{\text{BRST}}+1$ which is also a contradiction. Thus $c>2$.

Let $\deg(f)=0$ and $E_{\text{BRST}}=1$ then for

$$\begin{aligned}(\deg(f) + 2E_{\text{BRST}} + 1) + c(\deg(f) + E_{\text{BRST}} + 1) &= \hat{c}(\deg(f) + 2E_{\text{BRST}} + 1) \\ 3 + 3(2) &= 3(3)\end{aligned}$$

thus $c = 3$. Thus if $E_{\text{BRST}} > 0$ then $c \geq 3$. □

By construction c is one less that the number of polynomials needed to transmit a message. Thus by Theorem 11 the interleaving scheme requires four or more polynomials.

Theorem 12 *If $E_{\text{BRST}} > 0$ then $\hat{c} \leq c$.*

Proof. Assume that $\hat{c} > c$. We know that

$$\begin{aligned}\deg(f) + 2E_{\text{BRST}} + 1 + c(\deg(f) + E_{\text{BRST}} + 1) &= \hat{c}(\deg(f) + 2E_{\text{BRST}} + 1) \\ c(\deg(f) + E_{\text{BRST}} + 1) &= (\hat{c} - 1)(\deg(f) + 2E_{\text{BRST}} + 1) \\ c(\deg(f) + E_{\text{BRST}} + 1) &\geq c(\deg(f) + 2E_{\text{BRST}} + 1) \\ \deg(f) + E_{\text{BRST}} + 1 &\geq \deg(f) + 2E_{\text{BRST}} + 1\end{aligned}$$

This is a contradiction. Thus if $E_{\text{BRST}} > 1$ then $\hat{c} \leq c$ □

Theorem 13 *$c = \hat{c}$ if and only if $E_{\text{BRST}} \mid (\deg(f) + 1)$.*

Proof.

$$\begin{aligned}\deg(f) + 2E_{\text{BRST}} + 1 + c(\deg(f) + E_{\text{BRST}} + 1) &= c(\deg(f) + 2E_{\text{BRST}} + 1) \\ (c+1)\deg(f) + (c+2)E_{\text{BRST}} + c + 1 &= c\deg(f) + 2cE_{\text{BRST}} + c \\ \deg(f) + (2-c)E_{\text{BRST}} + 1 &= 0 \\ cE_{\text{BRST}} &= \deg(f) + 2E_{\text{BRST}} + 1 \\ c &= \frac{\deg(f) + 1}{E_{\text{BRST}}} + 2 \quad \square\end{aligned}$$

Corollary 4 *$c = \hat{c} = 3$ if and only if $E_{\text{BRST}} = \deg(f) + 1$.*

Proof. This follows from the previous theorem. □

Lemma 12 *$\hat{c}$ is the depth of the interleave.*

Proof. Observe that

$$\begin{aligned}\deg(f) + 2E_{\text{BRST}} + 1 + c(\deg(f) + E_{\text{BRST}} + 1) &= \hat{c}(\deg(f) + 2E_{\text{BRST}} + 1)\\ \hat{c} &= \frac{\deg(f) + 2E_{\text{BRST}} + 1 + c(\deg(f) + E_{\text{BRST}} + 1)}{\deg(f) + 2E_{\text{BRST}} + 1}\end{aligned}$$

by construction $c(\deg(f) + E_{\text{BRST}} + 1)$ is the number of evaluations of the second through to the last polynomial. Since $\deg(f) + 2E_{\text{BRST}} + 1$ is the length of the interleave $c(\deg(f) + E_{\text{BRST}} + 1)/(\deg(f) + 2E_{\text{BRST}} + 1)$ must be the depth of the polynomials $f^{[2]}$ to $f^{[m]}$. This is one less than the total depth. Observe that

$$c = 1 + \frac{c(\deg(f) + E_{\text{BRST}} + 1)}{\deg(f) + 2E_{\text{BRST}} + 1} \quad \square$$

Observe that

$$c(\deg(f) + E_{\text{BRST}} + 1) = (\hat{c} - 1)(\deg(f) + 2E_{\text{BRST}} + 1)$$

So the left hand side (LHS) is a multiple of $\deg(f) + 2E_{\text{BRST}} + 1$ and the right hand side (RHS) is a multiple of $\deg(f) + E_{\text{BRST}} + 1$. Since the $LHS = RHS$ we must have that the LHS and RHS is a common multiple of $\deg(f) + 2E_{\text{BRST}} + 1$ and $\deg(f) + E_{\text{BRST}} + 1$. Then the smallest value for LHS and RHS is the $\text{LCM}(\deg(f) + 2E_{\text{BRST}} + 1, \deg(f) + E_{\text{BRST}} + 1)$. This gives the following algorithm for computing c and $\hat{c}$.

Algorithm 8: Algorithm for Computing c and $\hat{c}$.

```
Input: $\deg(f), E_{\text{BRST}}$
Output: $c, \hat{c}$
1: $\text{lcm} \leftarrow \text{LCM}(\deg(f) + 2E_{\text{BRST}} + 1, \deg(f) + E_{\text{BRST}} + 1)$
2: $c \leftarrow \text{lcm}/(\deg(f) + E_{\text{BRST}} + 1)$
3: $\hat{c} \leftarrow (\text{lcm} + \deg(f) + 2E_{\text{BRST}} + 1)/(\deg(f) + 2E_{\text{BRST}} + 1)$
4: return $c, \hat{c}$
```

Proof.

$$\begin{aligned}\deg(f) + 2E_{\text{BRST}} + 1 + c(\deg(f) + E_{\text{BRST}} + 1) &= \hat{c}(\deg(f) + 2E_{\text{BRST}} + 1)\\ \deg(f) + 2E_{\text{BRST}} + 1 + \frac{\text{lcm}}{\deg(f) + E_{\text{BRST}} + 1}(\deg(f) + E_{\text{BRST}} + 1) &= \frac{(\text{lcm} + \deg(f) + 2E_{\text{BRST}} + 1)}{\deg(f) + 2E_{\text{BRST}} + 1}\\ &\quad (\deg(f) + 2E_{\text{BRST}} + 1)\\ \deg(f) + 2E_{\text{BRST}} + 1 + \text{lcm} &= \text{lcm} + \deg(f) + 2E_{\text{BRST}} + 1\end{aligned}$$

This shows that our algorithm works and that the c and $\hat{c}$ it computes is as small all possible. $\square$

6.4 Data Transmission

In the previous section we considered only burst errors. While this made the description of the interleaving scheme less cumbersome, it is only practical in the black box setting. In the black box setting we evaluate vectors so it is practical that an error affects every entry in the vector. Recall that for our algorithm to decode successfully a burst error must affect the complete Reed-Solomon codeword. This would mean, however, that all burst errors must affect the complete Reed-Solomon codeword. In example 4 the complete codeword is evaluations of the polynomial $f^{[1]}(u)$. In the data transmission setting, however, it is more likely that after transmission we have a mixture of burst and non-burst errors.

Suppose we can divide our error bound E into burst errors E_{BRST} and non-burst errors E_{NBRST} such that $E = E_{\text{BRST}} + E_{\text{NBRST}}$. We have shown that we can uniquely decode from $(c)(E_{\text{BRST}})$ fewer polynomial evaluations than the generalized Welch/Berlekamp algorithm. Thus we employ our interleaving strategy in order to transmit $(c)(E_{\text{BRST}})$ fewer polynomial evaluations. This suggests the following equation.

$$\deg(f)+2(E_{\text{BRST}}+E_{\text{NBRST}})+1+c(\deg(f)+E_{\text{BRST}}+2E_{\text{NBRST}}+1) = \hat{c}[\deg(f)+2(E_{\text{BRST}}+E_{\text{NBRST}})+1] \tag{6.3}$$

The value for c and $\hat{c}$ are computed as described by algorithm 8, with the following change: $\deg(f)+E+1$ is replaced by $\deg(f)+E_{\text{BRST}}+2E_{\text{NBRST}}+1$. The values of c and $\hat{c}$ depend on how E is splint into E_{BRST} and E_{NBRST} and a small change in the split of E can cause a significant change in the value of c and $\hat{c}$.

Given a message that is to be sent across a network that produces burst errors we need to decide how to encode the message. Let's assume that channel is likely to produce burst errors of length CBL. Recall that for the interleaving scheme any error that affects the complete Reed-Solomon codeword is burst error denoted by E_{BRST}. Depending on the interleave depth, $\hat{c}$, a channel burst of CBLmay produce one or more burst errors, E_{BRST}, and/or non-burst errors, E_{NBRST}. If we choose the interleave depth, $\hat{c}$, to the length CBL then channel bursts are likely to produce, at most one burst error and one non-burst error. See 6.6. So given CBL and a bound for the number of bursts, CBN, that occur we can compute values for E_{BRST} and E_{NBRST}.

In order to ensure that burst errors of length $\hat{c}$ always affects the complete Reed-Solomon codeword and at least one other codeword we place it in the middle of the interleave. For example the interleave in example 4 becomes

The evaluations $\beta_i^{[j]}$ in table 6.6 represents an erroneous evaluation. Let's assume that $\hat{c}$ = CBL = 3. Observe that no matter where the channel burst begins there is always one burst error (an error in a column that affect the complete Reed-Solomon codeword) and at most one non burst error (an error in a column that does not affect the complete Reed-Solomon

Table 6.5: Evaluated vectors interleaved for transmission

$\gamma_2^{[0]}$	$\gamma_2^{[1]}$	$\gamma_2^{[2]}$	$\gamma_2^{[3]}$	$\gamma_2^{[4]}$	$\gamma_2^{[5]}$	$\gamma_2^{[6]}$	$\gamma_2^{[7]}$	$\gamma_3^{[8]}$	$\gamma_3^{[9]}$	$\gamma_3^{[10]}$	$\gamma_3^{[11]}$
$\gamma_1^{[0]}$	$\gamma_1^{[1]}$	$\gamma_1^{[2]}$	$\gamma_1^{[3]}$	$\gamma_1^{[4]}$	$\gamma_1^{[5]}$	$\gamma_1^{[6]}$	$\gamma_1^{[7]}$	$\gamma_1^{[8]}$	$\gamma_1^{[9]}$	$\gamma_1^{[10]}$	$\gamma_1^{[11]}$
$\gamma_3^{[0]}$	$\gamma_3^{[1]}$	$\gamma_3^{[2]}$	$\gamma_3^{[3]}$	$\gamma_4^{[4]}$	$\gamma_4^{[5]}$	$\gamma_4^{[6]}$	$\gamma_4^{[7]}$	$\gamma_4^{[8]}$	$\gamma_4^{[9]}$	$\gamma_4^{[10]}$	$\gamma_4^{[11]}$

Table 6.6: Channel Burst Length

$\beta_2^{[0]}$	$\gamma_2^{[1]}$	$\gamma_2^{[2]}$	$\beta_2^{[3]}$	$\gamma_2^{[4]}$	$\beta_2^{[5]}$	$\gamma_2^{[6]}$	$\gamma_2^{[7]}$	$\gamma_3^{[8]}$	$\gamma_3^{[9]}$	$\gamma_3^{[10]}$	$\gamma_3^{[11]}$
$\beta_1^{[0]}$	$\gamma_1^{[1]}$	$\beta_1^{[2]}$	$\gamma_1^{[3]}$	$\gamma_1^{[4]}$	$\beta_1^{[5]}$	$\gamma_1^{[6]}$	$\gamma_1^{[7]}$	$\gamma_1^{[8]}$	$\gamma_1^{[9]}$	$\gamma_1^{[10]}$	$\gamma_1^{[11]}$
$\beta_3^{[0]}$	$\gamma_3^{[1]}$	$\beta_3^{[2]}$	$\gamma_3^{[3]}$	$\beta_4^{[4]}$	$\gamma_4^{[5]}$	$\gamma_4^{[6]}$	$\gamma_4^{[7]}$	$\gamma_4^{[8]}$	$\gamma_4^{[9]}$	$\gamma_4^{[10]}$	$\gamma_4^{[11]}$

codeword).

Next we describe how to compute a suitable value for $\deg(f)$. Observe

$$c(\deg(f) + E_{\text{BRST}} + 2E_{\text{NBRST}} + 1) = (\hat{c} - 1)(\deg(f) + 2(E_{\text{BRST}} + 2E_{\text{NBRST}}) + 1)$$
$$\implies \deg(f) = \frac{(2E_{\text{BRST}} + 2E_{\text{NBRST}} + 1)\hat{c} - (E_{\text{BRST}} + 2E_{\text{NBRST}} + 1)c - (2E_{\text{BRST}} + 2E_{\text{NBRST}} + 1)}{c - \hat{c} + 1}$$

Algorithm 9: Computing $\deg(f)$.

```
Input: CBL, CBN, ML
Output: $\deg(f)$
1: $\hat{c} \leftarrow$ CBL
2: $E_{\text{BRST}} \leftarrow$ CBN
3: $E_{\text{NBRST}} \leftarrow$ CBN
4: $c \leftarrow \hat{c}$
5: $\deg(f) \leftarrow (2E_{\text{BRST}} + 2E_{\text{NBRST}} + 1)\hat{c} - (E_{\text{BRST}} + 2E_{\text{NBRST}} + 1)c - (2E_{\text{BRST}} + 2E_{\text{NBRST}} + 1)$
6: return $\deg(f)$
```

The message we wish to send has a particular length, denoted by ML. Once we compute $\deg(f)$ we know that the message will be divided into blocks of size $(c+1)(\deg(f)+1)$. It is then possible that ML $\mod (c+1)(\deg(f)+1)$ is small relative to $(c+1)(\deg(f)+1)$. So we may wish to reduce the size of $\deg(f)$. The value $\deg(f)$ is a function of c. The part of the function we are concerned with is to the right of the vertical asymptote until it is no longer positive. The root of the function is

$$c = \frac{(\hat{c}-1)(2E_{\text{BRST}} + 2E_{\text{NBRST}} + 1)}{E_{\text{BRST}} + 2E_{\text{NBRST}} + 1}$$

To have an idea how many times we increment c we compute how far the root is from the asymptote

$$\frac{(\hat{c}-1)(2E_{\text{BRST}}+2E_{\text{NBRST}}+1)}{E_{\text{BRST}}+2E_{\text{NBRST}}+1}-(\hat{c}-1)=\frac{E_{\text{BRST}}(\hat{c}-1)}{E_{\text{BRST}}+2E_{\text{NBRST}}+1}$$

Any increment that produces an integer, for the value of $\deg(f)$ gives a different interleave.

6.5 Performance on a Simplified Gilbert Channel

Now that we have a description of our interleaving technique we will examine its performance on a specific channel model known to produce burst errors. The channel model we will use is known as the simplified Gilbert model.

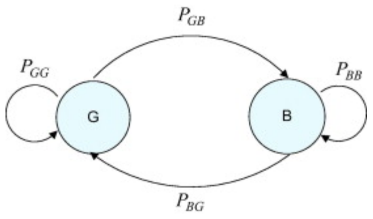

Figure 6.1: Gilbert Model for Burst Errors

The [Gilbert 1960] model is a two state Markov model. The two states are referred to as "good" and "bad". The probability that an error occurs while in the "good" state, P_G, is equal to zero. For the general Gilbert channel the probability, P_B, that an errors on the "bad" state may vary [Gilbert 1960; Mushkin and Bar-David 1989; Wolf 1998], where as in the Simplified Gilbert Channel [Yee and Weldon 1995] $P_B = 1$. Figure 6.1 is an illustration of the Gilbert model for a burst error channel. The probability of remaining in the "good" state is denoted by P_{GG}, while the probability of transitioning from the "good" state to the "bad" state is denoted by P_{GB}. The probability of remaining in the "bad" state is denoted by P_{BB}, while the probability of transitioning from the "bad" state to the "good" state is P_{BG}. Markov models are stochastic, so $P_{GB} = 1 - P_{GG}$ and $P_{BG} = 1 - P_{BB}$. The transition probability matrix is then

$$P = \begin{bmatrix} P_{GG} & 1 - P_{GG} \\ 1 - P_{BB} & P_{BB} \end{bmatrix} \tag{6.4}$$

The steady state probabilities of being in the "good" state and the "bad" state are

$$\pi_G = \tfrac{1-P_{BB}}{1-P_{BB}+1-P_{GG}} \quad \text{and} \quad \pi_B = \tfrac{1-P_{GG}}{1-P_{BB}+1-P_{GG}} \tag{6.5}$$

respectively. For the Simplified Gilbert Channel

$$\text{CBL} = \frac{1}{1 - P_{BB}}. \tag{6.6}$$

Let ρ be the average symbol error rate then

$$\rho = \frac{1 - P_{GG}}{1 - P_{BB} + 1 - P_{GG}}. \tag{6.7}$$

So that

$$\text{CBN} = \rho \text{ML}/\text{CBL}. \tag{6.8}$$

For our experiments we set $E_{\text{BRST}} = 2\lceil \text{CBN} \rceil$.

6.5.1 Double Interleaving

Given the message length, ML, we subdivide the message into interleaved blocks of depth $\hat{c}$. We may consider each of these blocks to be an interleaved Reed-Solomon codeword. We can then stack the blocks one on top the other and then transmit the columns from top to bottom. We call this strategy double interleaving. If $\hat{c} = 3$ then table 6.7 depicts the result of double interleaving.

Table 6.7: Evaluated vectors interleaved for transmission

$\gamma_2^{[0]}$	$\gamma_2^{[1]}$	$\gamma_2^{[2]}$	$\gamma_2^{[3]}$	$\gamma_2^{[4]}$	$\gamma_2^{[5]}$	$\gamma_2^{[6]}$	$\gamma_2^{[7]}$	$\gamma_3^{[8]}$	$\gamma_3^{[9]}$	$\gamma_3^{[10]}$	$\gamma_3^{[11]}$
$\gamma_1^{[0]}$	$\gamma_1^{[1]}$	$\gamma_1^{[2]}$	$\gamma_1^{[3]}$	$\gamma_1^{[4]}$	$\gamma_1^{[5]}$	$\gamma_1^{[6]}$	$\gamma_1^{[7]}$	$\gamma_1^{[8]}$	$\gamma_1^{[9]}$	$\gamma_1^{[10]}$	$\gamma_1^{[11]}$
$\gamma_3^{[0]}$	$\gamma_3^{[1]}$	$\gamma_3^{[2]}$	$\gamma_3^{[3]}$	$\gamma_4^{[4]}$	$\gamma_4^{[5]}$	$\gamma_4^{[6]}$	$\gamma_4^{[7]}$	$\gamma_4^{[8]}$	$\gamma_4^{[9]}$	$\gamma_4^{[10]}$	$\gamma_4^{[11]}$
$\gamma_6^{[0]}$	$\gamma_6^{[1]}$	$\gamma_6^{[2]}$	$\gamma_6^{[3]}$	$\gamma_6^{[4]}$	$\gamma_6^{[5]}$	$\gamma_6^{[6]}$	$\gamma_6^{[7]}$	$\gamma_7^{[8]}$	$\gamma_7^{[9]}$	$\gamma_7^{[10]}$	$\gamma_7^{[11]}$
$\gamma_5^{[0]}$	$\gamma_5^{[1]}$	$\gamma_5^{[2]}$	$\gamma_5^{[3]}$	$\gamma_5^{[4]}$	$\gamma_5^{[5]}$	$\gamma_5^{[6]}$	$\gamma_5^{[7]}$	$\gamma_5^{[8]}$	$\gamma_5^{[9]}$	$\gamma_5^{[10]}$	$\gamma_5^{[11]}$
$\gamma_7^{[0]}$	$\gamma_7^{[1]}$	$\gamma_7^{[2]}$	$\gamma_7^{[3]}$	$\gamma_8^{[4]}$	$\gamma_8^{[5]}$	$\gamma_8^{[6]}$	$\gamma_8^{[7]}$	$\gamma_8^{[8]}$	$\gamma_8^{[9]}$	$\gamma_8^{[10]}$	$\gamma_8^{[11]}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\gamma_{m-2}^{[0]}$	$\gamma_{m-2}^{[1]}$	$\gamma_{m-2}^{[2]}$	$\gamma_{m-2}^{[3]}$	$\gamma_{m-2}^{[4]}$	$\gamma_{m-2}^{[5]}$	$\gamma_{m-2}^{[6]}$	$\gamma_{m-2}^{[7]}$	$\gamma_{m-1}^{[8]}$	$\gamma_{m-1}^{[9]}$	$\gamma_{m-1}^{[10]}$	$\gamma_{m-1}^{[11]}$
$\gamma_{m-3}^{[0]}$	$\gamma_{m-3}^{[1]}$	$\gamma_{m-3}^{[2]}$	$\gamma_{m-3}^{[3]}$	$\gamma_{m-3}^{[4]}$	$\gamma_{m-3}^{[5]}$	$\gamma_{m-3}^{[6]}$	$\gamma_{m-3}^{[7]}$	$\gamma_{m-3}^{[8]}$	$\gamma_{m-3}^{[9]}$	$\gamma_{m-3}^{[10]}$	$\gamma_{m-3}^{[11]}$
$\gamma_{m-1}^{[0]}$	$\gamma_{m-1}^{[1]}$	$\gamma_{m-1}^{[2]}$	$\gamma_{m-1}^{[3]}$	$\gamma_m^{[4]}$	$\gamma_m^{[5]}$	$\gamma_m^{[6]}$	$\gamma_m^{[7]}$	$\gamma_m^{[8]}$	$\gamma_m^{[9]}$	$\gamma_m^{[10]}$	$\gamma_m^{[11]}$

Observe that there is no more than 2 transmissions between complete Reed-Solomon codewords. If E does indeed bound the number of bursts and they aren't many burst of length ≤ 2 we will be able to decode. One should also observe that we can increase the depth of the individual blocks. This would allow us to transmit fewer data. The draw back is, however, that there will be more separation between complete Reed-Solomon codewords.

6.5.2 Experimental Results

We designed an experiment in order to test our ideas. In the experiment we simulate sending messages across a Simplified Gilbert Channel and record our successes after 100 trials. We compare strategies that utilize double interleaving against ones that do utilize double interleaving. The results of out experiments are presented in the tables that follow.

Table 6.8: Experiment Parameters

P_{GG}	P_{BB}	analytic symbol error rate	actual symbol error rate	analytic mean burst length	actual mean burst length	analytic mean num of bursts	actual mean num of bursts
0.9999	0.9	0.000999	0.001647	10.00	12.45	0.15	0.20
	0.85	0.000666	0.000972	7.00	5.25	0.14	0.28
	0.8	0.000500	0.000595	5.00	4.50	0.15	0.20
0.999	0.9	0.009901	0.012626	10.00	10.72	1.50	1.78
	0.85	0.006623	0.007282	7.00	6.15	1.43	1.79
	0.8	0.004975	0.005681	5.00	4.67	1.50	1.84
0.99	0.9	0.090909	0.091898	10.00	9.85	13.75	14.11
	0.85	0.062500	0.060337	7.00	6.22	13.50	14.66
	0.8	0.047619	0.048505	5.00	5.08	14.40	14.44

Table 6.8 details how our simulations of the Gilbert Model compares to the theory. Table 6.8 is computed along with Table 6.9 and shows how the symbol error rate, mean burst length as well as the mean number of bursts computed compare to their respective theoretical values.

In Table 6.9, as in Section 6.3, we consider on only burst errors. The codeword is doubly interleaved and we choose our interleave depth to be the 3, which is the minimum. We compare there results with those of Table 6.10 where the only difference is that we do not doubly interleave the codewords. As expected this method does not work as well as double interleaving. The double interleave strategy extends the depth of the code, see Table 6.5, and allows for the decoding of longer bursts. This corroborated by observing the last column of Tables 6.9 and 6.10.

In Table 6.11 and 6.12 we again only consider burst error, but this time we do not choose the interleave depth to be the minimum. Instead we compute the ceiling of the channel busts length, see Equation (6.6), and use this as the depth of the interleave. In Table 6.11 we employ the double interleaving strategy, where as in Table 6.12 we do not. We again observe that double interleaving strategy is able to correct more burst as well as longer bursts.

Observe that the results in Table 6.11 while they aren't better than those in Table 6.9 they are quite close. Observe, however, that the difference between the message length and the codeword length is significantly smaller in Table 6.11 than in Table 6.9 We speculate the results in Table 6.9 are slightly better than those in Table 6.11 because using the minimum interleaving depths means that the complete Reed-Solomon codewords are always 2 evaluations apart, see Table 6.7. Where as in Table the values for $\hat{c}$ are $10, 7$, and 5. So the complete Reed-Solomon codewords are $9, 6$ and 4 evaluations apart respectively. Thus we are more likely to correct shorter bursts in Table 6.9 than in Table 6.11.

Table 6.9: Stack + Burst Only + Minimal Interleave

$c = \hat{c} = 3; \mathrm{ML} = 672; \mathrm{CL} = 1,512$

P_{GG}	P_{BB}	d_f	E_{BRST}	Pct of Correct Decoding	Total Num Bursts	Total Num Bursts Corrected	Max Burst Length	Max burst Length Corrected
0.9999	0.9	1	2	99.00	20.00	19.00	48.00	48.00
	0.85	1	2	96.00	28.00	22.00	11.00	10.00
	0.8	1	2	100.00	20.00	20.00	12.00	12.00
0.999	0.9	1	2	73.00	178.00	91.00	48.00	48.00
	0.85	1	2	69.00	179.00	86.00	25.00	25.00
	0.8	1	2	63.00	184.00	72.00	26.00	19.00
0.99	0.9	13	14	18.00	1411.00	198.00	72.00	42.00
	0.85	11	12	5.00	1466.00	45.00	45.00	24.00
	0.8	13	14	4.00	1444.00	54.00	29.00	19.00

Table 6.10: Not Stack + Burst Only + Minimal Interleave

$c = \hat{c} = 3; \text{ML} = 672; \text{CL} = 1,512$

P_{GG}	P_{BB}	d_f	E_{BRST}	Pct of Correct Decoding	Total Num Bursts	Total Num Bursts Corrected	Max Burst Length	Max burst Length Corrected
0.9999	0.9	1	2	80.00	29.00	3.00	48.00	5.00
	0.85	1	2	88.00	20.00	5.00	11.00	5.00
	0.8	1	2	87.00	24.00	7.00	12.00	2.00
0.999	0.9	1	2	22.00	178.00	8.00	48.00	5.00
	0.85	1	2	19.00	179.00	4.00	25.00	4.00
	0.8	1	2	29.00	191.00	17.00	15.00	5.00
0.99	0.9	13	14	0.00	1491.00	0.00	60.00	0.00
	0.85	11	12	0.00	1483.00	0.00	45.00	0.00
	0.8	13	14	0.00	1467.00	0.00	36.00	0.00

Table 6.11: Stack + Burst Only + Not Minimal Interleave

$c = \hat{c}; E_{\text{BRST}} = 2$

P_{GG}	P_{BB}	d_f	c	Message Length	Code Length	Pct of Correct Decoding	Total Num Bursts	Total Num Bursts Corrected	Max Burst Length	Max Burst Length Corrected
0.9999	0.9	15	10	528	600	97.00	11.00	7.00	26.00	26.00
	0.85	9	7	640	784	94.00	15.00	8.00	11.00	10.00
	0.8	5	5	684	950	96.00	17.00	12.00	12.00	12.00
0.999	0.9	15	10	528	600	74.00	72.00	28.00	41.00	41.00
	0.85	9	7	640	784	62.00	100.00	36.00	25.00	25.00
	0.8	5	5	684	950	56.00	119.00	37.00	26.00	26.00

Given the description of our experiment if we consider burst errors and non burst errors with the minimum interleave depth then d_f is less than zero. So we cannot consider this case with the current experiment parameters. We are however able to conduct our experiment in most cases when we set the interleave depth to be the ceiling of the channel burst length. See Table 6.13 and Table 6.14. As expected these Tables contain the best results as the account for burst errors as well a non burst errors. Recall that we expect channel burst that are the channel burst length to produce one burst error and one one burst error. See Tables 6.5 and 6.6.

Table 6.12: Not Stack + Burst Only + Not Minimal Interleave

$c = \hat{c}; E_{\text{BRST}} = 2$

P_{GG}	P_{BB}	d_f	c	Message Length	Code Length	Pct of Correct Decoding	Total Num Bursts	Total Num Bursts Corrected	Max Burst Length	Max burst Length Corrected
0.9999	0.9	15	10	528	600	90.00	11.00	0.00	26.00	0.00
	0.85	9	7	640	784	89.00	15.00	2.00	11.00	10.00
	0.8	5	5	684	950	89.00	17.00	5.00	12.00	7.00
0.999	0.9	15	10	528	600	56.00	72.00	7.00	41.00	20.00
	0.85	9	7	640	784	45.00	98.00	8.00	25.00	7.00
	0.8	5	5	684	950	41.00	117.00	16.00	26.00	9.00

Table 6.13: Stack + Not Burst Only + Not Minimal Interleave

$c = \hat{c}; E_{\text{BRST}} = E_{\text{NBRST}} = 2$

P_{GG}	P_{BB}	d_f	c	Message Length	Code Length	Pct of Correct Decoding	Total Num Bursts	Total Num Bursts Corrected	Max Burst Length	Max Burst Length Corrected
0.9999	0.9	11	10	660	1000	100.00	18.00	18.00	48.00	48.00
	0.85	5	7	672	1372	100.00	19.00	19.00	11.00	11.00
0.999	0.9	11	10	660	1000	100.00	125.00	125.00	45.00	45.00
	0.85	5	7	672	1372	98.00	156.00	146.00	25.00	25.00
0.99	0.85	35	7	576	1176	98.00	1151.00	1109.00	45.00	45.00

Table 6.14: Not Stack + Not Burst Only + Not Minimal Interleave

$c = \hat{c}; E_{\text{BRST}} = E_{\text{NBRST}} = 2$

P_{GG}	P_{BB}	d_f	c	Message Length	Code Length	Pct of Correct Decoding	Total Num Bursts	Total Num Bursts Corrected	Max Burst Length	Max Burst Length Corrected
0.9999	0.9	11	10	660	1000	98.00	17.00	13.00	48.00	26.00
	0.85	5	7	672	1372	100.00	19.00	19.00	11.00	11.00
0.999	0.9	11	10	660	1000	85.00	125.00	86.00	45.00	34.00
	0.85	5	7	672	1372	85.00	156.00	104.00	25.00	24.00
0.99	0.85	35	7	576	1176	87.00	1151.00	937.00	45.00	45.00

6.6 Summary

In this chapter we present an application of polynomial vector recovery. The application we present is polynomial vector recovery as an interleaved Reed-Solomon Code. We view the coefficients of the polynomials as components of a message that is sent over a channel known to produce burst errors. Rather than sending evaluation of one polynomial sequentially we interleave the evaluations in such away that we have maximum separation between evaluations of the same polynomial.

Burst error correcting codes have better error correcting capabilities than non burst error correcting codes. The same is true for polynomial vector recovery when errors occur in bursts. We use the idea of Chapter 5 to design an interleaving scheme that would allow us to decode from fewer evaluations than would be necessary if we did not use interleaving. We showed that the minimum interleaving depth is 3.

Recall that in our setting the burst error correcting code comprises of a single complete Reed-Solomon codeword. It is known that combining multiple burst error correcting codes also improves the error correcting capabilities of the code. This fact motivated us to fix the depth of the interleave and subdivide the message into subdivision of that depth. Each individual subdivision comprises of depth plus one many polynomials. We can the then stack all the subdivisions to create a deeper interleave that has as many complete Reed-Solomon codewords as there are subdivisions. We tested our idea on a simplified Gilbert Channel and present some preliminary results.

Chapter 7

Conclusion

In this thesis we begin with a review of the [Boyer and Kaltofen 2014] algorithm for solving full rank parametric linear systems with error correction. The algorithm works by interpolating the solution from scalar linear systems, given by a black box procedure. Using techniques from Coding Theory the algorithm works even if some of the scalar systems are erroneous. By erroneous we mean that the evaluated solution, of the parametric system, is not the solution of the scalar system returned by the black box. The algorithm simultaneously computes the solution and an error locator polynomial. We then present two early termination strategies that can we used to compute the solution from fewer evaluations. We then combine our two early termination strategies to create a general early termination strategy for solving full rank parametric linear systems with error correction.

We then show that the problem of recovering a black box vector of rational functions from its evaluations, where some evaluations may be erroneous, is a special case of the [Boyer and Kaltofen 2014] algorithm for parametric linear system solving with errors. Thus our early terminations strategies apply. To ensure uniqueness of the rational function we consider the common monic denominator of minimal degree. For early termination if roots of the common denominator, are allowed then we need extra information about the numerator of the solutions at places were the black box indicates that the common denominator is zero.

The thesis continues with the development of theory that leads to an algorithm for finding low degree solutions for rank deficient parametric linear systems with error correction. Like the algorithm for full rank systems the rank deficient algorithm works by interpolating the solution from scalar linear systems given by a black box. The algorithm works even some scalar systems returned by the black box are erroneous. By erroneous we mean that one or more of the evaluated solutions of the parametric linear system is not a solution to the scalar system returned by the black box. Unlike the algorithm for the full rank case, our algorithms for the rank deficient case does not compute an error locator polynomial and low degree solution simultaneously.

Instead, our algorithm for the rank deficient case first computes the error locator polynomial, then removes the erroneous systems returned by the black box. Finally the algorithm computes a low degree solution from the error free systems.

Note that a scalar system returned by the black box can have the same solution as the evaluated parametric linear system only if their respective matrices have the same null space. We are able to remove all erroneous systems by first classifying all errors as one of two types:

1. Matrix Errors, are errors that occur because the null space of the scalar matrix returned by the black box is not equal to the null space of the evaluated matrix for the parametric linear system. Such an error indicates one or more of the relationships among the columns in the evaluated matrix of the parametric system does not exist among the columns in the scalar matrix returned by the black box. We show that we can locate all such errors by examining a canonical basis for the null space of scalar systems returned by the black box.

2. Right Side Vector Errors, are errors that occur when the null space of the scalar matrix returned by the black box is equal to the null space of the evaluated matrix for the parametric linear system, but the scalar right side vector returned by the black box is such that one or more of the solutions of the evaluated parametric system is not a solution to the scalar system returned by the black box. A key observation is that all error in the full rank case can be classified as Right Side Vector Errors. Thus we are able to identify all such errors by first finding a set linearly independent columns in the matrix of the parametric system and using only those columns with the algorithm for the full rank case.

Once we have identified and removed all erroneous systems returned by the black box we are then able to use the error free systems to compute a low degree solution.

The thesis ends with a discussion of solving a black box vector of polynomials and its connection to interleaved Reed-Solomon Codes for decoding messages transmitted over channels that cause burst errors. For the polynomial recovery problem we consider an error model where if the black box returns an error then then a particular entry in the vector is bad. This entry is a part of what he term the complete Reed-Solomon codeword. Note that a model where an error affects all entries satisfies the condition that a particular entry is always affected. We show that for this error model we can solve a smaller system than is required for the error model where an error may not affect a particular entry in the vector. We further show that there is no need to assume that the errors are random and that the solution and error locations can still be found simultaneously.

In Coding Theory code interleaving is a technique used correcting errors when errors occur in bursts. If the vector being recovered is 1 dimensional, that is there is only one polynomial, then the algorithm is the [Gemmell and Sudan 1992] description of the [Welch and Berlekamp 1986] decoder for [Reed and Solomon 1960] codes. Polynomial vector recovery then can be viewed as an interleaving of Reed-Solomon codes. Given that we can reduce the size of the system we solve when a particular entry in the vector is bad we devise a strategy to take advantage of this on a burst error channel. The idea is to be able to send fewer data than is required by the Generalized Welch/Berlekamp Algorithm across the channel while maintaining a high probability of decoding. We give a description of the strategy, discuss some limitations and finally test our ideas on a Simplified [Gilbert 1960] Channel, which is a common model used in Coding Theory for testing burst error correcting codes.

Chapter 8

Future Work

In Chapter 2 we presented early termination strategies for parametric linear system solving with error correction for full rank systems. Where as, in Chapter 4 we presented the first algorithm for parametric linear system solving with error correction for rank deficient systems. The algorithm that we prove is exponential in the size of the matrix. We later conjectured an algorithm, 5 that is linear in the size of the matrix, but we do no prove that it is always correct. I would like to supply a proof of the correctness our conjectured algorithm. I also plan to investigate whether the early termination strategies for full rank systems apply in the rank deficient case. If not, then what improvements are possible for our initial algorithm for rank deficient systems.

In Chapter 3 we showed that rational vector recovery with error correction is specialization of parametric linear system solving with error correction for full rank systems. Two natural question are:

1. Are there any more application for the full rank algorithm.

2. What are some applications for the rank deficient algorithm.

I plan to investigate these questions along with other questions that may result from the quest for their answers.

In Chapter 5 we discussed polynomial vector recovery with burst error and then in Chapter 6 we showed how it could be used as a Reed-Solomon burst error correcting code. There exists extensive research on error correcting codes and error correcting codes has many practical use. I think it would be interesting to see how our findings are related to what is already known about burst error correcting Reed-Solomon codes. Does our work add any insight to the ongoing research of Coding Theory?

It may be possible that error correction can be used as an algorithmic paradigm, like Divide and Conquer or Early Termination. Say in the execution of program there are some computations which take very long, while others are fast. Perhaps we can place dummy values for the output of the slow computations and use error correction to find their values later and cheaper. This is a interesting idea that I believe deserves some attention.

Coding Theory and Cryptography are closely related, see [Kiayias and Yung 2002] and [Kiayias and Yung 2008]. It may be possible to a public key encryption protocol based on the idea of error correction. I intend to devote some time trying to find such a protocol.

REFERENCES

Bleichenbacher, Daniel, Kiayias, Aggelos, and Yung, Moti. Decoding of interleaved reed solomon codes over noisy data. In *International Colloquium on Automata, Languages, and Programming*, pages 97–108. Springer, 2003.

Boyer, Brice B. and Kaltofen, Erich L.. Numerical linear system solving with parametric entries by error correction. In Verschelde, Jan and Watt, Stephen M., editors, *SNC'14 Proc. 2014 Internat. Workshop on Symbolic-Numeric Comput.*, pages 33–38, New York, N. Y., 2014. Association for Computing Machinery. URL: `http://www.math.ncsu.edu/~kaltofen/bibliography/14/BoKa14.pdf`.

Cabay, Stanley. Exact solution of linear equations. In *Proceedings of the Second ACM Symposium on Symbolic and Algebraic Manipulation*, SYMSAC '71, pages 392–398, New York, NY, USA, 1971. ACM. URL `http://doi.acm.org/10.1145/800204.806310`.

Gemmell, Peter and Sudan, Madhu. Highly resilient correctors for polynomials. *Information processing letters*, 43(4):169–174, 1992.

Gilbert, Edgar N. Capacity of a burst-noise channel. *Bell Labs Technical Journal*, 39(5): 1253–1265, 1960.

Kaltofen, Erich and Pernet, Clément. Cauchy interpolation with errors in the values, December 2013.

Kaltofen, Erich and Yang, Zhengfeng. Sparse multivariate function recovery from values with noise and outlier errors. In Kauers, Manuel, editor, *ISSAC 2013 Proc. 38th Internat. Symp. Symbolic Algebraic Comput.*, pages 219–226, New York, N. Y., 2013. Association for Computing Machinery. URL: `http://www.math.ncsu.edu/~kaltofen/bibliography/13/KaYa13.pdf`.

Kaltofen, Erich L. and Yang, Zhengfeng. Sparse multivariate function recovery with a high error rate in evaluations. In Nabeshima, Katsusuke, editor, *ISSAC 2014 Proc. 39th Internat. Symp. Symbolic Algebraic Comput.*, pages 280–287, New York, N. Y., 2014. Association for Computing Machinery. URL: `http://www.math.ncsu.edu/~kaltofen/bibliography/14/KaYa14.pdf`.

Kaltofen, Erich L., Pernet, Clément, Storjohann, Arne, and Waddell, Cleveland. Early termination in parametric linear system solving and rational function vector recovery with error correction. In *Proceedings of the International Symposium on Symbolic and Algebraic Computation*, pages 237–244. ACM, 2017.

Kiayias, Aggelos and Yung, Moti. Cryptographic hardness based on the decoding of reed-solomon codes. In *International Colloquium on Automata, Languages, and Programming*, pages 232–243. Springer, 2002.

Kiayias, Aggelos and Yung, Moti. Cryptographic hardness based on the decoding of reed–solomon codes. *IEEE Transactions on Information Theory*, 54(6):2752–2769, 2008.

Leung, Kon and Welch, L. Erasure decoding in burst-error channels. *IEEE Transactions on Information Theory*, 27(2):160–167, 1981.

McClellan, M. T. The exact solution of systems of linear equations with polynomial coefficients. *J. ACM*, 20:563–588, 1973.

Mushkin, Mordechai and Bar-David, Israel. Capacity and coding for the gilbert-elliott channels. *IEEE Transactions on Information Theory*, 35(6):1277–1290, 1989.

Olesh, Zach and Storjohann, Arne. The vector rational function reconstruction problems. In *Proc. Waterloo Workshop on Computer Algebra: devoted to the 60th birthday of Sergei Abramov (WWCA)*, pages 137–149, 2007.

Olshevsky, V. and Shokrollahi, M. A. A displacement approach to decoding algebraic codes. In *Algorithms for Structured Matrices: Theory and Applications*, volume 323, pages 265–292. American Math. Soc., 2003.

Pernet, Clément. *High Performance Algebraic Reliable Computing.* Habilitation thesis, Univ. Joseph Fourier (Grenoble 1), November 2014.

Reed, Irving S and Solomon, Gustave. Polynomial codes over certain finite fields. *Journal of the society for industrial and applied mathematics*, 8(2):300–304, 1960.

Schmidt, Georg, Sidorenko, Vladimir, and Bossert, Martin. Collaborative decoding of interleaved Reed-Solomon codes and concatenated code designs. *IEEE Transactions on Information Theory*, 55:2991 – 3012, 2009.

Storjohann, Arne and Villard, Gilles. Computing the rank and a small nullspace basis of a polynomial matrix. In *Proceedings of the 2005 international symposium on Symbolic and algebraic computation*, pages 309–316. ACM, 2005.

Welch, L. R. and Berlekamp, E. R. Error correction of algebraic block codes. US Patent 4,633470, 1986. See `http://patft.uspto.gov/`.

Wolf, JK. Ecc performance of interleaved rs codes with burst errors. *IEEE Transactions on Magnetics*, 34(1):75–79, 1998.

Yee, James R and Weldon, Edward J. Evaluation of the performance of error-correcting codes on a gilbert channel. *IEEE transactions on Communications*, 43(8):2316–2323, 1995.

APPENDIX

Appendix A

Maple Programs

Start of code

```
#Implementation of the algorithms for Numerical Linear System Solving
With Parametric Entries By Error Correction

ParmLinSolvWithErrors := module()
  export  KP_Lin_Solv #Solves linear equations using the KP bound.
         ,COS_Lin_Solv #Solves linear eqauations using the COS bound.
         ;
  local  Lin_Solv;
  option  package;

  interface(rtablesize=infinity);

  Lin_Solv := proc({BB:=Ex1 #Bloack Box matrix.
                 , L::integer:=0 #The number of evaluations.
                 ,df::integer:=0 #bound for numerator degree.
                 ,dg::integer:=0 #bound for denominator degree.
                 , R::integer:=0 #rank drop bound.
                 , E::integer:=0 #error bound.
                 })


  #------------------------------------------------------
  # Output: x, the solution to the system A(u)x = b(u).
```

```
#-----------------------------------------------------


  local xi          #Set of evalution points.
        ,A           #Black box matrix.
        ,b           #Black box vector.
        ,m           #Row dimension of A.
        ,n           #Column dimension of A.
        ,AOfXi       #Evaluated matrix.
        ,gOfU        #Denominator of the solution.
        ,vectPhiOfU  #Vector of unknown coefficient phi.
        ,psiOfU      #Polynomial in unknown coefficient psi.
        ,vectOfEqns  #Vector of equations in unknowns phi and psi.
        ,listOfEqns  #Vector of equations converted to list.
        ,coefMat     #Coefficient matrix of linear system.
        ,sol         #Nullspace space of coefficient matrix.
        ,x           #Solution in the form f^[i](u)/g(u).
        ,i           #Index.
        ,j           #Index.
        ,k           #Accumulation of ranks drops.
        ,err         #Error loactor polynomial.
        ,xLambda     #Solution before factoring error polynomial.
        ,solVect     #Solution vector.
        ,solIndex    #Index.
        ,scaleFact   #Scaling factor that makes g(u) monic.
        ,solCheck    #Check that the solution computed is correct.
        ,gaussElimMat#get low degree solution.
        ,Ell         #The number of evaluations minus the rank drops.
        ;

#------------------------------
# Initialization of variables
#------------------------------

  A:=BB:-Umat('u');
  b:=BB:-Uvect('u');
  m:=BB:-Umatdims()[1];
  n:=BB:-Umatdims()[2];
  Ell := L;
  xi:={seq(-i,i=1..floor(L/2)),seq(j,j=0..ceil(L/2)-1)};
  k:=0;
  err:=1;
  scaleFact:=1;

#---------------------------------------------------------------
# Create a vector of polynomials in the unknown coefficent phi
#---------------------------------------------------------------

  vectPhiOfU := Vector(n,i->sum(phi[i][j]*u^j,j=0..df+E));

#---------------------------------------------------
# Create polynomial in the unknown coefficient psi
#---------------------------------------------------



  psiOfU := sum(psi[j]*u^j,j=0..dg+E);

#--------------------------------------
# Make big vector of linear equations
#--------------------------------------

  vectOfEqns := eval(LinearAlgebra:-Add(-1*psiOfU*BB:-Uvect(xi[1])[1],
              BB:-Umat(xi[1])[1].vectPhiOfU), u=xi[1]);

  for i from 2 to Ell do
    vectOfEqns := ArrayTools:-Concatenate(1,vectOfEqns,
    eval(LinearAlgebra:-Add(-1*psiOfU*BB:-Uvect(xi[i]),
    BB:-Umat(xi[i]).vectPhiOfU), u=xi[i]));
  od;

#--------------------------------------------------------------------
# Convert vector to list and set each linear equation equal to zero
#--------------------------------------------------------------------

  listOfEqns := convert(vectOfEqns,list);
  for i from 1 to nops(listOfEqns) do
    listOfEqns[i] := vectOfEqns[i] = 0;
  od;

#----------------------------------------
# Generate coefficient matrix and solve
#----------------------------------------

  coefMat := LinearAlgebra:-GenerateMatrix(listOfEqns,[seq(psi[dg+E-j],
             j=0..dg+E), seq(seq(phi[i][df+E-j],j=0..df+E),i=1..n)],
             augmented=false);

  sol := LinearAlgebra:-NullSpace(coefMat[1]);

  if sol = {} then
    error "No Soltion found, one or more bounds must me increased";
  else

    gaussElimMat := LinearAlgebra:-GaussianElimination(LinearAlgebra:-
                    Transpose((convert(convert(sol,list),Matrix))));
    solVect := LinearAlgebra:-Row(gaussElimMat,nops(sol));
  fi;

  gOfU := add(solVect[i]*u^(dg+E+1-i),i=1..dg+E+1);

#---------------------------------------------------------------------
# check whether or not g(u) is zero. If it is then go up further in
# row reduced matrix until g(u) is no longer 0
#---------------------------------------------------------------------

  if gOfU = 0 then
    solIndex := 1;
  fi;
  while gOfU = 0 do
    solVect := LinearAlgebra:-Row(gaussElimMat,nops(sol)-solIndex);
    gOfU := add(solVect[i]*u^(dg+E+1-i),i=1..dg+E+1);
    solIndex := solIndex + 1;
  od;

#---------------------------------------------------------------
# Solution vector before factoring the error locator polynomial
#---------------------------------------------------------------

  xLambda :=[Vector(n,[seq(add(solVect[dg+E+1+j+k*(df+E+1)]*u^(df+E+1-j),
  j=1..df+E+1),k=0..n-1)]), gOfU]; # Convert solution to polynomial form.

#--------------------------------------------
# Factoring out the error locator polynomial
#--------------------------------------------

  err := gcd(xLambda[1][1],gOfU);
  for i from 2 to n do
    err := gcd(xLambda[1][i],err);
  od;

  gOfU := simplify(gOfU/err);

  #--------------------------------------------------------------
  # If g(u) is not monic then we divide all coeffcients by the
  # leading coefficient of g(u)
  #--------------------------------------------------------------

  if coeff(gOfU,u,degree(gOfU,u)) <> 1 then
    scaleFact := coeff(gOfU,u,degree(gOfU,u));
    gOfU := gOfU/coeff(gOfU,u,degree(gOfU,u));
  fi;

  x := [simplify(err^(-1)*xLambda[1]/scaleFact),gOfU];

 #-------------------------------------------
 #Check that the solution computed is correct
 #-------------------------------------------

  solCheck := simplify(A.(err^(-1)*xLambda[1]/scaleFact) - gOfU*b);
  for i from 1 to m do
    if solCheck[i] <> 0 then
      error "The computed solution is incorrect. Look for bugs";
    fi;
  od;
  return x, factor(err);

end proc; #End of linSolv procedure.


KP_Lin_Solv :=
proc({ BBMat:=Ex1
      ,d_f::integer:=0 #bound for numerator degree.
      ,d_g::integer:=0 #bound for denominator degree.
      ,RBd::integer:=0 #rank drop bound.
      ,EBd::integer:=0 #error bound.
     })
#---------------------------------------------
# Output: x, the solution to equation (5)
#---------------------------------------------

  local LKP #The number of evalutions.
       ;

#------------------------------
# Initialization of variables
#------------------------------

  LKP:=d_f+d_g+RBd+2*EBd+1;

#----------------------------
#Get and return solution
#----------------------------

  return Lin_Solv(BB=BBMat, L=LKP, df=d_f, dg=d_g, R=RBd, E=EBd);
  end proc; #End of KP_Lin_Solv procedure.


  COS_Lin_Solv :=
  proc({ BBMat:=Ex1
        ,d_f::integer:=0 #bound for numerator degree.
        ,d_g::integer:=0 #bound for denominator degree.
        ,dA::integer:=0 #bound max degree polynomial in A.
        ,db::integer:=0 #boun max degree polynomila in b.
        ,RBd::integer:=0 #rank drop bound
        ,EBd::integer:=0 #error bound.
        })

  #-----------------------------------------------------
  # Output: x, the solution to the system A(u)x = b(u).
  #------------------------------------------------------

    local LCOS        #The number of evalutions.
         ;

  #------------------------------
  # Initialization of variables
  #------------------------------

    LCOS:=max(dA+d_f,db+d_g)+2*EBd+RBd+1;

  #----------------------------
  #Get and return solution
  #----------------------------

    return Lin_Solv(BB=BBMat, L=LCOS, df=d_f, dg=d_g, R=RBd, E=EBd);
  end proc; #End of COS_Lin_Solv procedure.

end module:
```

End of code

Start of code

```
### Sample balck box

Ex11 := module()
  export Umat
        ,Umatdims
        ,Uvect
      ;
  local A
       ,A_at_xi
       ,b
       ,b_at_xi
       ;
  A := Matrix(2,2,[[u^9+3*u^8+13*u^7-12*u^6-u^4+3*u^3+7*u^2+15*u-7,
       u^9+7*u^8-2*u^7-u^6-u^5+u^4+4*u^3+10*u^2+2*u+1],
       [-5*u^9-9*u^8-12*u^7-8*u^6-14*u^5+23*u^4+21*u^3-6*u^2+14*u-6,
       -4*u^9-6*u^8-11*u^7-19*u^6-u^5+18*u^4+9*u^3+9*u^2+3*u]]);

  Umat := proc(xi)
    local A_at_xi;
    A_at_xi:= map(x -> subs('u'=xi,x), A);
    if xi = 5 then A_at_xi[1,1] := 0; fi; # error at 5
    if xi = 4 then A_at_xi[1,2] := -1; fi; #error at 4
    return A_at_xi;
  end proc;

  b := Vector(2,[2*u^10+10*u^9+6*u^8+2*u^7-12*u^6+u^5+5*u^4+16*u^3+
       14*u^2+7*u-7, -9*u^10-16*u^9-21*u^8-32*u^7+4*u^6+17*u^5+29*
       u^4+23*u^3-u^2+8*u-6]);

  Uvect := proc(xi)
    local b_at_xi;
    b_at_xi := map(x -> subs('u'=xi,x),b);
    return b_at_xi;
  end proc;
  Umatdims := proc()
    return 2,2;
  end proc;
end module: # end of Ex11
```

End of code

Start of code

```
### Implementation of the algorithm for Rank Defficeint Linear Systems
### This implementation first solve the homegenous system.

RankDefParmLinSolv := module()

  export ,SING_Lin_Solv
         ;

  local Rank_Def_Lin_Solv
        ;

  option package
         ;

  interface(rtablesize=infinity);

  Rank_Def_Lin_Solv := proc({BB:=BB2 #Bloack Box matrix.
                  , L::integer:=0 #The number of evaluations.
                  ,df::integer:=0 #bound numerator degree .
                  ,dg::integer:=0 #bound denominator degree.
                  ,dA::integer:=0 #bound max degree polynomial in A.
                  ,db::integer:=0 #bound max degree polynomial in b.
                  , R::integer:=0 #rank drop bound.
                  , E::integer:=0 #error bound.
                  })

  #-----------------------------------------------------
  # Output: x, the solution to the system A(u)x = b(u).
```

```
#-------------------------------------------------------


  local xi           #Set of evalution points.
        ,Ahatsupbrell  #matrix returned by the black box.
        ,bhatsupbrell  #right side vector retruned by the black box.
        ,Ab           #Augmentation of Asupbrell and bsupbrell.
        ,RREFAb       #Reduced Row Echelon Form of Ab.
        ,zeros        #bhatsupbrell is zero.
        ,A            #Black box matrix.
        ,b            #Black box vector.
        ,m            #Row dimension of A.
        ,n            #Column dimension of A.
        ,Phi_U        #Polynomial in unknown coefficients phi_i
        ,Omega_U      #Polynomail in unknown coefficients omega_i
        ,Psi_U        #Polynomial in unknown coefficients psi_i
        ,Lambda_U     #Polynomial in unkowwn coefficients lambda_i
        ,gOfU         #Denominator of the solution.
        ,vectPhiOfU   #Vector of unknown coefficient phi.
        ,psiOfU       #Polynomial in unknown coefficient psi.
        ,vectOfEqns   #Vector of equations in unknowns phi and psi.
        ,listOfEqns   #Vector of equations converted to list.
        ,coefMat      #Coefficient matrix of linear system.
        ,sol          #Nullspace space of coefficient matrix.
        ,i            #Index.
        ,j            #Index.
        ,k            #Accumulation of ranks drops.
        ,l            #Index.
        ,ii           #Index.
        ,jj           #Index.
        ,kk           #Index.
        ,err          #Error loactor polynomial.
        ,xLambda      #Solution before factoring error polynomial.
        ,solVect      #Solution vector.
        ,solIndex     #Index.
        ,scaleFact    #Scaling factor that makes g(u) monic.
        ,solCheck     #Check that the solution computed is correct.
        ,gaussElimMat#Gaussian elimination.
```

```
,error_set   #The set of all erroroneus evaluations.
,phi_index   #Index for the Phi polynomials.
,omega_index #Index for the Omega polynomials
,psi_index   #Index for the Psi polynomial.
,lam_index   #Index the lambda polynomial.
,hat_sys      #Ahat and bhat.
,lambda_u    #error locator polynomial.
,Err         #A replacement for E
,Phi_list    #List of solutions for Phi
,Omega_list    #List of solutions for Omega
,SqRREFAb    #Pad RREFAb with zeros until square
,ToF         #True or False?
,bzero       #Is right side vector zero?
,colval      #Only zero column
,hat_list    #list of augemented hatted systems from black box
,non_dep_set #Set of non dependent columns
,pre_piv_col #The previous independent column
,dep_set     #Set of dependent columns
,uni_set     #Universal set. First n integers
,zero_col #The non-zero column
,omega_ord_list#The order of the phi's for gaussian elimination.
,old_omega_ord_list #older order of phi for gaussian elimination
,nonzero_omega #The set of nonzero phi's for homogenous solution.
,zero_omega
,d_fpart     #numerator degree bound for particular solution.
,zero_col_Ahat #The columns that evaluate to zero.
,bhat_zero_set #The set of evaluations with bhat equal to zero
,zero_col_list #A list of lists. [[eval pt, zero_col]]
,zero_list     #Make the zero col list
,fact          #Factor to remove from Phi
,div_counter
,first_element #Omegas that have been first
,perm_counter  #number of permuatation checked
,temp
,d_Omega       #null space degree bound
;
```

```
#------------------------------
# Initialization of variables
#------------------------------

  zeros := {};
  A := BB:-Umat;
  b := BB:-Uvect;
  m:=BB:-mat_dim()[1];
  n:=BB:-mat_dim()[2];
  error_set := {};
  d_Omega := n*dA;
  xi := {seq(-i,i=1..floor(L/2)),seq(j,j=0..ceil(L/2)-1)};
  k := 0;
  err := 1;
  Lambda_U := 1;
  scaleFact:=1;
  hat_list := [];
  uni_set := {seq(i, i = 1 .. n)};
  bhat_zero_set := {};
  zero_col_list := [];
  zero_list := [];
  fact := 1;


#-------------------------------------------------------------------
# List of points to interpolate to find a matrix row equivalent to A.
#-------------------------------------------------------------------
  Omega_U := Vector(n, i->sum(omega[i][omega_index]*u^omega_index,
             omega_index = 0 .. d_Omega + E));


  hat_sys := BB:-hat_lin_sys(xi[1]);
  Ahatsupbrell := hat_sys[1];
  vectOfEqns := eval(Ahatsupbrell.Omega_U, u = xi[1]);

  for i from 2 to nops(xi) do
    hat_sys := BB:-hat_lin_sys(xi[i]);
    Ahatsupbrell := hat_sys[1];
    vectOfEqns := ArrayTools:-Concatenate(1, vectOfEqns,
                  eval(Ahatsupbrell.Omega_U ,u = xi[i]));
  od;

  listOfEqns := convert(vectOfEqns, list);
  omega_ord_list := [seq(i, i = 1..n)];
  first_element := [1];

  perm_counter := 0;

  while perm_counter < (n - 1) do

    coefMat := LinearAlgebra:-GenerateMatrix(listOfEqns,
               [seq(seq(omega[i][d_Omega + E - jj],jj = 0..d_Omega + E),
               i = omega_ord_list)], augmented=false);

    sol := LinearAlgebra:-NullSpace(coefMat[1]);
    if sol = {} then
      break;
    else

      gaussElimMat := LinearAlgebra:-GaussianElimination(LinearAlgebra:-
                      Transpose(convert(convert(sol,list),Matrix)));

      solVect := LinearAlgebra:-Row(gaussElimMat, nops(sol));
    fi;

    Omega_list := Vector(n,[seq(add(solVect[j+k*(d_Omega+E+1)]*
                 u^(d_Omega+E+1-j), j = 1..d_Omega + E + 1), k=0..n-1)]);

    err := gcd(Omega_list[1],Omega_list[1]);

    for i from 2 to n do
      err := gcd(Omega_list[i], err);
    od;
    Lambda_U := Lambda_U * err;

    if Omega_list[1] <> 0 then
      break;
    fi;

    for i from 2 to (n - 1) do

      if Omega_list[i] <> 0 then
        temp := Omega_list[1];
        Omega_list[1] := Omega_list[perm_counter + 1];
        Omega_list[i] := temp;
      fi;

      for j from 1 to nops(first_element) do
        if i = first_element[j] then
          omega_ord_list := [seq(omega_ord_list[k], k = 1 .. i)];
          i := n - 1;
        fi;
      od;

    od;

    perm_counter := perm_counter + 1;

  od;

  error_set := {RealDomain:-solve(Lambda_U)};

  xi := xi minus error_set;

  Err := E - nops(error_set);
  ### remove error many more evaluations.
  for i from 1 to nops(error_set) do
    zeros := {op(zeros), xi[i]};
  od;
```

```
    xi := xi minus zeros;

########################################################################

  #---------------------------------------------------------------------
  # List of points to interpolate to find a matrix row equivalent to A.
  #---------------------------------------------------------------------
    Phi_U := Vector(n, i->sum(phi[i][phi_index]*u^phi_index,
                  phi_index = 0 .. df + E));

    Psi_U := sum(psi[psi_index]*u^psi_index,psi_index=0..dg + E);

    hat_sys := BB:-hat_lin_sys(xi[1]);
    Ahatsupbrell := hat_sys[1];
    bhatsupbrell := hat_sys[2];

    zero_col_Ahat := {};
    for j from 1 to n do  ### identify the colums that evaluated to zero
      ToF := LinearAlgebra:-Equal(LinearAlgebra:-Column(Ahatsupbrell, j),
             LinearAlgebra:-ZeroVector(m));
      if ToF = true then
        zero_col_Ahat := {op(zero_col_Ahat), j};
      fi;
    od;

    RREFAb := LinearAlgebra:-
              ReducedRowEchelonForm(<Ahatsupbrell|bhatsupbrell>);

    SqRREFAb := RREFAb[1..m,1..n];
    if m < n then ### Padd B with zero rows until it is square.
      for i from 1 to n-m do
        SqRREFAb := ArrayTools:-Concatenate(1, SqRREFAb, LinearAlgebra:-
                  Transpose(LinearAlgebra:-ZeroVector(n)));
      od;
    fi;

    vectOfEqns := eval(LinearAlgebra:-Add(-1*Psi_U*bhatsupbrell,
               Ahatsupbrell.Phi_U),u=xi[1]);

    ToF := LinearAlgebra:-
           Equal(bhatsupbrell, LinearAlgebra:-ZeroVector(m));
    ### Right side evaluates to zero.
    if ToF = true then
      bhat_zero_set := {op(bhat_zero_set), xi[1]};
      if nops(zero_col_Ahat) <> 0 then
         zero_list := [op(zero_list), xi[1]];
         for i from 1 to nops(zero_Ahat_list) do
           zero_list := [op(zero_list), zero_col_Ahat[i]];
         od;
         zero_col_list := [op(zero_col_list), zero_list];
         zero_list := [];
      fi;
    fi;


  #---------------------------------------------------------------------
  # Added equations to get a particular soluion.
  #---------------------------------------------------------------------

  ### Find the zero columns

    for j from 1 to n do
      if LinearAlgebra:-Equal(LinearAlgebra:-Column(SqRREFAb, j),
        Vector(m)) then
        non_dep_set:= {op(non_dep_set), j};
      fi;
    od;

  ### Find independent columns

    i := 1;
    pre_piv_col := 1;

    while i <= m do
```

```
    for j from pre_piv_col to n do
      if SqRREFAb[i,j] <> 0 then
        non_dep_set := {op(non_dep_set), j};
        pre_piv_col := j;
        break
      fi;
    od;
    i := i + 1;
    if j = n then
      i := m + 1;
    fi;
  od:

### Find dependent columns

  dep_set := uni_set minus non_dep_set;
  dep_set := dep_set minus zero_col_Ahat;
### Remove Columns that evaluate to zero from the dependent set.
  zero_col_Ahat := {};
  for i from 1 to nops(dep_set) do
    vectOfEqns := ArrayTools:-Concatenate(1, vectOfEqns,
                  eval(Phi_U[dep_set[i]], u = xi[1]));
  od;

### empty sets
  dep_set := {};
  non_dep_set := {};

  for k from 2 to nops(xi) do
    hat_sys := BB:-hat_lin_sys(xi[k]);
    Ahatsupbrell := hat_sys[1];
    bhatsupbrell := hat_sys[2];
     ### identify the columns that evaluated to zero.
    for j from 1 to n do
      ToF := LinearAlgebra:-Equal(LinearAlgebra:-
             Column(Ahatsupbrell, j),
      LinearAlgebra:-ZeroVector(m));
      if ToF = true then
        zero_col_Ahat := {op(zero_col_Ahat), j};
      fi;
    od;

### Right side evaluates to zero.
    ToF := LinearAlgebra:-Equal(bhatsupbrell, LinearAlgebra:-
           ZeroVector(m));
    if ToF = true then
      bhat_zero_set := {op(bhat_zero_set), xi[k]};
      if nops(zero_col_Ahat) <> 0 then
        zero_list := [op(zero_list), xi[k]];
        for i from 1 to nops(zero_Ahat_list) do
          zero_list := [op(zero_list), zero_col_Ahat[i]];
        od;
        zero_col_list := [op(zero_col_list), zero_list];
        zero_list := [];
      fi;
    fi;

    RREFAb := LinearAlgebra:-
              ReducedRowEchelonForm(<Ahatsupbrell|bhatsupbrell>);
    SqRREFAb := RREFAb[1..m,1..n];

    if m < n then
      for j from 1 to n-m do
        SqRREFAb := ArrayTools:-Concatenate(1, SqRREFAb, LinearAlgebra:-
                    Transpose(LinearAlgebra:-ZeroVector(n)));
      od;
    fi;

    vectOfEqns := ArrayTools:-
                  Concatenate(1, vectOfEqns, eval(LinearAlgebra:-
                  Add(-1*Psi_U*bhatsupbrell, Ahatsupbrell.Phi_U),
                  u=xi[k]));

#---------------------------------------------------------------------
```

```
# Added equations to get a particular soluion.
#--------------------------------------------------------------------


    zero_col := 0;

    for j from 1 to n do
      if SqRREFAb[j,j] = 0  then
        ToF := LinearAlgebra:-
               Equal(LinearAlgebra:-Column(SqRREFAb, j),
               LinearAlgebra:-ZeroVector(n));
        if ToF = true then
          zero_col := zero_col + 1;
          colval := j;
        fi;
      fi;
      if zero_col = 1 and j = n then
        vectOfEqns := ArrayTools:-Concatenate(1, vectOfEqns,
                      eval(Phi_U[colval], u = xi[k]));
      fi;
    od;

### Find the zero columns

    for j from 1 to n do
      if LinearAlgebra:-Equal(LinearAlgebra:-Column(SqRREFAb, j),
         Vector(m)) then
        non_dep_set:= {op(non_dep_set), j};
      fi;
    od;

### Find independent columns

    i := 1;
    pre_piv_col := 1;

    while i <= m do
      for j from pre_piv_col to n do
```

```
        if SqRREFAb[i,j] <> 0 then
          non_dep_set := {op(non_dep_set), j};
          pre_piv_col := j;
          break
        fi;
      od;
      i := i + 1;
      if j = n then
        i := m + 1;
      fi;
    od;

### Find dependent columns

    dep_set := uni_set minus non_dep_set;
    zero_col_Ahat := {};

### add equations

    for i from 1 to nops(dep_set) do
      vectOfEqns := ArrayTools:-Concatenate(1, vectOfEqns,
      eval(Phi_U[dep_set[i]], u = xi[k]));
    od;

### empty sets

    dep_set := {};
    non_dep_set := {};

  od;

  listOfEqns := convert(vectOfEqns, list);

  coefMat := LinearAlgebra:-GenerateMatrix(listOfEqns,[seq(psi[dg+E-ii],
             ii=0..dg+E), seq(seq(phi[i][df+E-jj], jj=0..df+E),i=1..n)],
             augmented=false);
```

```
sol := LinearAlgebra:-NullSpace(coefMat[1]);

if sol = {} then
  error "No Solution found.";

else
  gaussElimMat := LinearAlgebra:-GaussianElimination(LinearAlgebra:-
                Transpose(convert(convert(sol,list),Matrix)));
  solVect := LinearAlgebra:-Row(gaussElimMat, nops(sol));
fi;

Psi_U := add(solVect[kk]*u^(dg + E - kk + 1), kk = 1..dg + E + 1);

Phi_list := Vector(n,[seq(add(solVect[dg+E+1+j+k*(df+E+1)]*u^(df+E+1-j),
           j=1..df+E+1), k=0..n-1)]);

for i from 1 to nops(zero_col_list) do
  div_counter := 0;
  ToF:=divide((Psi_U), (u-zero_col_list[i, 1]));
  if ToF = true then
    div_counter := div_counter + 1;
  fi;
  for j from 2 to nops(zero_col_list[i]) do
    ToF:=divide((Phi_list[zero_col_list[i,j]]), (u-zero_col_list[i,1]));
    if ToF = true then
      div_counter := div_counter + 1;
    fi;
  od;
  if div_counter = nops(zero_col_list[i]) then
    Psi_U := quo((Psi_U), (u - zero_col_list[i, 1]), u);
    for j from 2 to nops(zero_col_list[i]) do
      Phi_list[zero_col_list[i,j]]:=quo((Phi_list[zero_col_list[i,j]]),
     (u - zero_col_list[i,1]), u);
    od;
  fi;
od;
```

```
    err := gcd(Phi_list[1], Psi_U);

    for i from 2 to n do
      err := gcd(Phi_list[i], err);
    od;

    Lambda_U := expand(Lambda_U * err);

    error_set := {RealDomain:-solve(err)};

    xi := xi minus error_set;

    Err := Err - nops(error_set);
  ### remove extra evaluations that were added for errors.
    for i from 1 to 2*Err do
      zeros := {op(zeros), xi[i]};
    od;

    xi := xi minus zeros; ### removal done here.

######################################################################

  #--------------------------------------------------------------------
  # List of points to interpolate to find a matrix row equivalent to A.
  #--------------------------------------------------------------------
    Phi_U := Vector(n, i->sum(phi[i][phi_index]*u^phi_index,
             phi_index=0..df));

    Psi_U := sum(psi[psi_index]*u^psi_index,psi_index=0..dg);


    hat_sys := BB:-hat_lin_sys(xi[1]);
    Ahatsupbrell := hat_sys[1];
    bhatsupbrell := hat_sys[2];

    vectOfEqns := eval(LinearAlgebra:-Add(-1*Psi_U*bhatsupbrell,
                  Ahatsupbrell.Phi_U),u=xi[1]);
    for i from 2 to nops(xi) do
      hat_sys := BB:-hat_lin_sys(xi[i]);
      Ahatsupbrell := hat_sys[1];
      bhatsupbrell := hat_sys[2];

      vectOfEqns := ArrayTools:-Concatenate(1, vectOfEqns,
                  eval(LinearAlgebra:-Add(-1*Psi_U*bhatsupbrell,
                  Ahatsupbrell.Phi_U),u=xi[i]));

    od;

    listOfEqns := convert(vectOfEqns, list);


    coefMat := LinearAlgebra:-GenerateMatrix(listOfEqns,[seq(psi[dg-ii],
               ii=0..dg), seq(seq(phi[i][df-jj], jj=0..df),i=1..n)],
               augmented=false);
    sol := LinearAlgebra:-NullSpace(coefMat[1]);
    if sol = {} then
      error "No Solution found.";
    else

      gaussElimMat := LinearAlgebra:-GaussianElimination(LinearAlgebra:-
                      Transpose(convert(convert(sol,list),Matrix)));
      solVect := LinearAlgebra:-Row(gaussElimMat, nops(sol));
    fi;

    Psi_U := add(solVect[kk]*u^(dg - kk + 1), kk = 1..dg + 1);

  #------------------------------------------------------------------
  # check whether or not Psi(u) is zero. If it is then go up further
  # in row reduced matrix until Psi(u) is no longer 0
  #------------------------------------------------------------------

    if Psi_U = 0 then
      solIndex := 1;
```

```
 fi;
 while Psi_U = 0 do
   solVect := LinearAlgebra:-Row(gaussElimMat,nops(sol)-solIndex);
   Psi_U := add(solVect[i]*u^(dg + 1 - i),i = 1..dg + 1);
   solIndex := solIndex + 1;
 od;

 Phi_list := Vector(n,[seq(add(solVect[dg+1+j+k*(df+1)]*u^(df+1-j),
             j=1..df + 1),k=0..n-1)]);

#----------------------------------------------------------------
# If Psi_U in not monic then we divide all coefficients by the
# leading coefficient of Psi_U
#----------------------------------------------------------------

 if coeff(Psi_U, u, degree(Psi_U, u)) <> 1 then
   scaleFact := coeff(Psi_U, u, degree(Psi_U, u));
   Psi_U := Psi_U/coeff(Psi_U, u, degree(Psi_U,u));
 fi;

 Phi_list := Phi_list/scaleFact; ### Scale the Phi vector

#------------------------------------------
#Check that the solution computed is correct
#------------------------------------------

 solCheck := simplify(A.Phi_list - Psi_U*b);

 for i from 1 to m do
   if solCheck[i] <> 0 then
     error "The computed solution is incorrect. Look for bugs";
   fi;
 od;



 return Psi_U, Phi_list, factor(Lambda_U);
```

```
end proc; ### End of Lin_Solv procedure.

SING_Lin_Solv :=
proc({ BBMat:=BB2
      ,d_f::integer:=0 #bound numerator degree.
      ,d_g::integer:=0 #bound denominator degree.
      ,d_A::integer:=0 #bound max degree polynomial in A.
      ,d_b::integer:=0 #bound max degree polynomila in b.
      ,RBd::integer:=0 #rank drop bound.
      ,EBd::integer:=0 #error bound.
      })

#-----------------------------------------------------
# Output: x, the solution to the system A(u)x = b(u).
#-----------------------------------------------------

  local LSING #The number of evalutions.
        ,d_Omega
        ,n
       ;

#------------------------------
# Initialization of variables
#------------------------------
  n:=BBMat:-mat_dim()[2];
  d_Omega := n*d_A;
  LSING := d_A + d_Omega + RBd + 2*EBd + 1;

#----------------------------
#Get and return solution
#----------------------------

  return Rank_Def_Lin_Solv(BB=BBMat, L=LSING, df=d_f, dg=d_g, db=d_b,
                       dA=d_A, R=RBd, E=EBd);
end proc; #End of COS_Lin_Solv procedure.
end module:
```

End of code

Start of code

```
### Sample black box for the rank deficient case.

BB13 := module() # Rank deficient

  export hat_lin_sys ### Accepts xi_ell and returns Ahat and bhat.
        ,mat_dim     ### Returns the dimention of A.
        ,Umat
        ,Uvect
        ;

  local A            ### The matrix.
       ,b            ### The right side vector.
       ,Aug          ### A and b augmented.
       ,Aug_at_xi    ### Evaluation of Aug.
       ,RR_Aug_at_xi ### Some row reduction(s) applied to Aug_at_xi.
       ,Ahat         ### The scaler matrix returned by the black box.
       ,bhat         ### scaler right side vector returned by black box.
       ,m            ### The row dimension of A.
       ,n            ### The column dimension of A.
       ,row_num      ### A random fuction to produce random row number.
       ,row_mul      ### A random funcion to produce random multiplier.
       ;
  A := Matrix(2,2,[['u+1,'u'+3],['u'^2+3*'u'+ 2, 'u'^2+5*'u'+6]]);
  b := Vector(2,[2*'u'^3+12*'u'^2+20*'u'+10,
              2*'u'^4+16*'u'^3+44*'u'^2+50*'u'+20]);
  Aug := linalg:-augment(A,b);
  Aug := convert(Aug, Matrix);
  m := LinearAlgebra:-RowDimension(A);
  n := LinearAlgebra:-ColumnDimension(A);

  Umat := A;
```

```
Uvect := b;

hat_lin_sys := proc(xi)    ### Procedure that returns Ahat and bhat.

  local Aug_at_xi
       ,RR_Aug_at_xi
       ,Ahat
       ,bhat
       ,row_num := rand(1..n)
       ,row_mul := rand(1..11)
       ;

  Aug_at_xi:= map(x -> subs('u'=xi,x), Aug);

 ### Introduction of ERRORS  to the system. ###

  if xi = 3 then
    Aug_at_xi[1,2] := 0;
    Aug_at_xi[2,2] := 0;
    Aug_at_xi[1,1] := 0;
    Aug_at_xi[2,1] := 0;

 fi; # error at 3.

 if xi = 2 then
   Aug_at_xi[1,2] := 0;
   Aug_at_xi[2,2] := 0;
 fi; #error at 2.

 if xi = 1 then
   Aug_at_xi[1,3] := 10;
   Aug_at_xi[2,3] := 10;
 fi; #error at 1.

### End of ERRORS.###
### Apply some what random row operation to Aug_at_xi. ###
```

```
    RR_Aug_at_xi := LinearAlgebra:-RowOperation(Aug_at_xi,[row_num(),
                    row_num()], row_mul());

  ### End of Row Operation.

    Ahat := LinearAlgebra:-SubMatrix(RR_Aug_at_xi,[1..m],[1..n]);
    bhat := LinearAlgebra:-Column(RR_Aug_at_xi, n+1);

    return Ahat
          ,bhat
        ;

  end proc;  ### End of precdure that returns Ahat and bhat.

  mat_dim := proc() ### Procedure that returns the dimension of A.

    return m, n;

  end proc;  ### End of procedure that returns the dimension of A.

end module: ### End of the black box module.
```

End of code

Start code

```
total_num_bursts_corrected := 0;
total_num_bursts := 0;
max_burst_length_corrected := 0;
max_num_bursts_corrected := 0;
max_num_bursts := 0;
max_burst_length := 0;
success_count := 0;
iival := 1;
error_count := 0;
```

End code

Start of code

```
### Implementation of our interleave h a Gilbert model.
### The interleaving is not stacked. We assume there are as many
### non burst errors as there are burst errors.
### Created: 01/28/19
### Programmer: Cleveland Waddell

Sim_Pol_Vec_Slv_Bst_Ers := module()

  export KW_Interleave_Parameters ### Retrieves user input and computes the
                                  ### polynomial degree and bounds for the
                                  ### number of burst and non burst errors
         ;


  local Pol_Vec_Rec_Slv ### Recovers the polynomials
        ,path_maple_files
        ,exp_par_list
        ,exp_header_list
        ;

##################### For Checkpointing ##############################
  global total_num_bursts_corrected
        ,total_num_bursts
        ,max_burst_length_corrected
        ,max_num_bursts_corrected
        ,max_num_bursts
        ,max_burst_length
        ,success_count
        ,iival
        ,error_count
        ,exp_par_tbl_mat
        ,sta_bol_min_tbl_mat
        ,not_sta_bol_min_tbl_mat
       ;
```

```
option package
       ;

interface(rtablesize=infinity);

################ Experiment Parameters table intialization #######
exp_par_list :=[seq(i, i = 1..6)];
exp_header_list := [seq(i, i = 1 .. 15)];


Pol_Vec_Rec_Slv := proc(Ebrst, NonEbrst, d_f, P_GG, P_BB, c_hat, c,
                        Stack, Burst_Only, minimal_interleave, rho,
                        bl, ml)

  local i ### Index
       ,j ### Index
       ,k ### Index
       ,ii ### Index
       ,jj ### Index
       ,kk ### Index
       ,list_of_eqns
       ,psi_of_u
       ,E
       ,df
       ,m ### Matrix row index
       ,n ### Matrix columns index
       ,xi
       ,coef_mat
       ,sol
       ,gauss_elim_mat
       ,sol_vect
       ,sol_index
       ,sol_times_lambda_vect
       ,depth ### interleave depth
       ,L ### the number of evaluation
```

```
     ,err
     ,scale_fact
     ,x
     ,g_of_u
     ,success ### True or false
     ,good_state ### True or false
     ,P_GB_fraction
     ,P_BG_fraction
     ,P_GB
     ,P_BG
     ,GB_denom_list
     ,BG_denom_list
     ,prob_GG
     ,prob_BB
     ,error_value
     ,GB_transition
     ,BG_transition
     ,no_solution_count
     ,burst_counter
     ,transed_BG ### Transitioned from the Bad State to the Good State
     ,transed_GB ### Transitioned from the Good State to the Bad State
     ,avg_num_bursts
     ,burst_length
     ,mean_burst_length
     ,max_trial_burst_length
     ,num_trial_bursts
     ,mean_num_bursts_corrected
     ,total_pseudo_sols
     ,pseudo_count ### counts the number of entries in the false vector
                   ### that agrees with the original vector
     ,runs
     ,row ### table row index
     ,tbl_id
     ,exp_tbl_id
     ;

runs := 100;
```

```
    randomize(ithprime(20419));
    pseudo_count := 0;
    total_pseudo_sols := 0;
    burst_length := 0;
    no_solution_count := 0;
    error_value := rand(2..100);
    P_GB := 1 - P_GG;
    P_BG := 1 - P_BB;
    prob_GG := P_GG;
    prob_BB := P_BB;
    good_state := true; #cw 01/30/19
    transed_GB := false;
    transed_BG := false;
    m := LinearAlgebra:-RowDimension(message:-pol_vector);
    depth := LinearAlgebra:-RowDimension(message:-phi_interleave);
    L := LinearAlgebra:-ColumnDimension(message:-phi_interleave);
    xi:={seq(-i, i = 1 .. floor(L/2)), seq(j, j = 0 .. ceil(L/2) - 1)};
    E := Ebrst + NonEbrst;
    df := d_f;
    psi_of_u := sum(psi[j]*u^j, j = 0 .. E);
    P_GB_fraction := convert(P_GB, rational);
    GB_transition := rand(1 .. denom(P_GB_fraction));
    P_BG_fraction := convert(P_BG, rational);
    BG_transition := rand(1 .. denom(P_BG_fraction));
    path_maple_files := "../Runs/Maple_Files/";

########## Read Checkpoint file and Matrices to store table values ########

    if Stack=true and Burst_Only=true and minimal_interleave=true and
    P_GG = 0.9999 then
      read "./Checkpoint/sbomi1_1_checkpoint.mpl";
      read cat(path_maple_files,"exp_par_tbl_3rows.mpl");
      read cat(path_maple_files,"sta_bol_min_tbl_3rows.mpl");
    fi;

    if Stack=true and Burst_Only=true and minimal_interleave=true and
    P_GG = 0.999 then
```

```
      read "./Checkpoint/sbomi1_2_checkpoint.mpl";
      read cat(path_maple_files,"exp_par_tbl_3rows.mpl");
      read cat(path_maple_files,"sta_bol_min_tbl_3rows.mpl");
    fi;

    if Stack=true and Burst_Only=true and minimal_interleave=true and
    P_GG = 0.99 then
      read "./Checkpoint/sbomi1_3_checkpoint.mpl";
      read cat(path_maple_files,"exp_par_tbl_3rows.mpl");
      read cat(path_maple_files,"sta_bol_min_tbl_3rows.mpl");
    fi;

    if Stack=false and Burst_Only=true and minimal_interleave=true and
    P_GG = 0.9999 then
      read "./Checkpoint/nsbomi4_1_checkpoint.mpl";
      read cat(path_maple_files,"not_sta_bol_min_tbl_3rows.mpl");
    fi;

    if Stack=false and Burst_Only=true and minimal_interleave=true and
    P_GG = 0.999 then
      read "./Checkpoint/nsbomi4_2_checkpoint.mpl";
      read cat(path_maple_files,"not_sta_bol_min_tbl_3rows.mpl");
    fi;

    if Stack=false and Burst_Only=true and minimal_interleave=true and
    P_GG = 0.99 then
      read "./Checkpoint/nsbomi4_3_checkpoint.mpl";
      read cat(path_maple_files,"not_sta_bol_min_tbl_3rows.mpl");
    fi;

######################################################################

    for ii from iival to runs do
      list_of_eqns := [];
      err := 1;
      scale_fact := 1;
      burst_counter := 0;
      burst_length := 0;
      max_trial_burst_length := 0;
      num_trial_bursts := 0;

      if Stack then

        for j from 1 to L do

          for i from 1 to depth do

###################### Transition Check ###########################

            if good_state and GB_transition()<=numer(P_GB_fraction) then
              transed_GB := true;
              good_state := false;
              burst_counter := burst_counter + 1;
              transed_BG := false;
            elif good_state then
              transed_GB := false;
              transed_BG := false;

            elif not good_state and BG_transition()<=
            numer(P_BG_fraction) then
              transed_BG := true; ### tansitioned from bad to good
              good_state := true;
              transed_GB := false;
            else
              transed_BG := false;
              transed_GB := false;
            fi;

#####################################################################

            if good_state = true then
              list_of_eqns := [op(list_of_eqns),
              eval(-1*psi_of_u*message:-pol_interleave[i, j] +
              message:-phi_interleave[i, j], u = xi[j])];
            else
              list_of_eqns := [op(list_of_eqns),
              eval(-1*error_value()*psi_of_u +
              message:-phi_interleave[i, j], u = xi[j])];
              error_count := error_count + 1;
            fi;

############ What happens if in the bad state #####################

            if not good_state then
              burst_length := burst_length + 1;
            fi;

### What if we transition from the bad state back to the good state? #

            if good_state and transed_BG and max_burst_length <
               burst_length then
              max_burst_length := burst_length;
            fi;

            if good_state and transed_BG and max_trial_burst_length <
               burst_length then
              max_trial_burst_length := burst_length;
            fi;

            if not good_state and j=L and i=depth and max_burst_length <
               burst_length then
              max_burst_length := burst_length;
            fi;

            if not good_state and j=L and i=depth and
            max_trial_burst_length < burst_length then
              max_trial_burst_length := burst_length;
            fi;

            if good_state and transed_BG then
              burst_length := 0;
```

```
        fi;


######### Did we transition from the good to the bad state? #############
          od:

        od:

      else  ### Not Stack

        jj := 1;

        for kk from 1 to depth/c_hat do

          for j from 1 to L do

            for i from jj to kk*c_hat  do

              if good_state and GB_transition()<=numer(P_GB_fraction) then
                transed_GB := true;
                good_state := false;
                burst_counter := burst_counter + 1;
                transed_BG := false;
              elif good_state then
                transed_GB := false;
                transed_BG := false;
              elif  not good_state and BG_transition() <=
              numer(P_BG_fraction) then
                transed_BG := true; ### tansitioned from bad to good
                good_state := true;
                transed_GB := false;
              else
                transed_BG := false;
                transed_GB := false;
              fi;

              if good_state = true then
```

```
                list_of_eqns := [op(list_of_eqns), eval(-1*psi_of_u*
                                 message:-pol_interleave[i, j] +
                                 message:-phi_interleave[i, j], u=xi[j])];
              else
                list_of_eqns := [op(list_of_eqns), eval(-1*error_value()*
                                 psi_of_u + message:-phi_interleave[i, j],
                                 u = xi[j])];
                error_count := error_count + 1;
              fi;

              if not good_state then
                burst_length := burst_length + 1;
              fi;

#### What if we transition from the bad state back to the good state? ##


            if good_state and transed_BG and max_burst_length <
               burst_length then
              max_burst_length := burst_length;
            fi;

            if good_state and transed_BG and max_trial_burst_length <
               burst_length then
              max_trial_burst_length := burst_length;
            fi;

            if not good_state and j=L and i=depth and max_burst_length <
               burst_length then
              max_burst_length := burst_length;
            fi;

            if not good_state and j = L and i = depth and
            max_trial_burst_length < burst_length then
              max_trial_burst_length := burst_length;
            fi;
```

```
        if good_state and transed_BG then
          burst_length := 0;
        fi;

######## Did we transition from the good to the bad state? ################

        od:

      od:
      jj := jj + c_hat;
    od:

  fi;

  if max_num_bursts < burst_counter then
    max_num_bursts := burst_counter;
  fi;

  total_num_bursts := total_num_bursts + burst_counter;

  coef_mat := LinearAlgebra:-GenerateMatrix(list_of_eqns,
              [seq(psi[E - j], j = 0 .. E), seq(seq(phi[i]
              [df + E - j],j = 0 .. df + E), i =1 .. m)],
              augmented=false);

  sol := LinearAlgebra:-NullSpace(coef_mat[1]);

  if nops(sol) >= 1 then

    gauss_elim_mat := LinearAlgebra:-GaussianElimination
                      (LinearAlgebra:-Transpose((convert
                      (convert(sol, list), Matrix))));

    sol_vect := LinearAlgebra:-Row(gauss_elim_mat, nops(sol));

  fi;
```

```
    g_of_u := add(sol_vect[i]*u^(E + 1 - i), i = 1 .. E + 1);

#------------------------------------------------------------------
# check whether or not g(u) is zero. If it is then go up further
# in row reduced matrix until g(u) is no longer 0
#------------------------------------------------------------------

    if g_of_u = 0 then
      sol_index := 1;
    fi;
    while g_of_u = 0 do
      sol_vect := LinearAlgebra:-Row(gauss_elim_mat, nops(sol) -
                  sol_index);
      g_of_u := add(sol_vect[i]*u^(E + 1 - i), i = 1 .. E + 1);
      sol_index := sol_index + 1;
    od;

#-------------------------------------------------------------------
# Solution vector before factoring the error locator polynomial
#-------------------------------------------------------------------

    sol_times_lambda_vect := [Vector(m, [seq(add(sol_vect[E+1+j+k*
                             (df+E+1)]*u^(df+E+1-j),j=1..df
                             +E+1), k=0..m-1)]), g_of_u];

#-----------------------------------------
# Factoring out the error locator polynomial
#-----------------------------------------

    err := gcd(sol_times_lambda_vect[1][1], g_of_u);
    for i from 2 to m do
      err := gcd(sol_times_lambda_vect[1][i], err);
    od;

    g_of_u := simplify(g_of_u/err);

  #-----------------------------------------------------------------
```

```
    # If g(u) is not monic then we divide all coeffcients by the
    # leading coefficient of g(u)
    #-----------------------------------------------------------------

      if coeff(g_of_u, u, degree(g_of_u, u)) <> 1 then
        scale_fact := coeff(g_of_u, u, degree(g_of_u, u));
        g_of_u := g_of_u/coeff(g_of_u, u, degree(g_of_u, u));
      fi;

      x := [simplify(err^(-1)*sol_times_lambda_vect[1]/scale_fact),
           g_of_u];

      success := ArrayTools:-IsEqual(simplify(x[1]-message:-pol_vector),
                 Vector(m, 0));


####################### Flatten if statements ###########################

      total_num_bursts := total_num_bursts + burst_counter;

      if success then
        success_count := (success_count + 1);
        total_num_bursts_corrected := total_num_bursts_corrected +
        burst_counter;

        for i from 1 to nops(x[1]) while (degree(x[1][i]) =
        degree(message:-pol_vector[i])) do
          pseudo_count := pseudo_count + 1;
        od:

      fi;

      pseudo_count := 0;

      if success and max_burst_length_corrected<max_trial_burst_length then
        max_burst_length_corrected := max_trial_burst_length;
      fi;
      if success and  max_num_bursts_corrected < burst_counter then
        max_num_bursts_corrected := burst_counter;
      fi;

    fi;

      iival := iival + 1;

      if Stack=true and Burst_Only=true and minimal_interleave=true and
      P_GG = 0.9999 then
        save iival, success_count, total_num_bursts_corrected,
        total_num_bursts, max_burst_length_corrected, error_count,
        max_num_bursts_corrected, max_num_bursts, max_burst_length,
        prob_GG, prob_BB, "./Checkpoint/sbomi1_1_checkpoint.mpl":
      fi;

      if Stack=true and Burst_Only=true and minimal_interleave=true and
      P_GG = 0.999 then
        save iival, success_count, total_num_bursts_corrected,
        total_num_bursts, max_burst_length_corrected, error_count,
        max_num_bursts_corrected, max_num_bursts, max_burst_length,
        prob_GG, prob_BB, "./Checkpoint/sbomi1_2_checkpoint.mpl":
      fi;

      if Stack=true and Burst_Only=true and minimal_interleave=true and
      P_GG = 0.99 then
        save iival, success_count, total_num_bursts_corrected,
        total_num_bursts, max_burst_length_corrected, error_count,
        max_num_bursts_corrected, max_num_bursts, max_burst_length,
        prob_GG, prob_BB, "./Checkpoint/sbomi1_3_checkpoint.mpl":
      fi;

      if Stack=false and Burst_Only=true and minimal_interleave=true and
      P_GG = 0.9999 then
        save iival, success_count, total_num_bursts_corrected,
        total_num_bursts, max_burst_length_corrected, error_count,
        max_num_bursts_corrected, max_num_bursts, max_burst_length,
        prob_GG, prob_BB, "./Checkpoint/nsbomi4_1_checkpoint.mpl":
      fi;

      if Stack=false and Burst_Only=true and minimal_interleave=true and
      P_GG = 0.999 then
        save iival, success_count, total_num_bursts_corrected,
        total_num_bursts, max_burst_length_corrected, error_count,
        max_num_bursts_corrected, max_num_bursts, max_burst_length,
        prob_GG, prob_BB, "./Checkpoint/nsbomi4_2_checkpoint.mpl":
      fi;

      if Stack=false and Burst_Only=true and minimal_interleave=true and
      P_GG = 0.99 then
        save iival, success_count, total_num_bursts_corrected,
        total_num_bursts, max_burst_length_corrected, error_count,
        max_num_bursts_corrected, max_num_bursts, max_burst_length,
        prob_GG, prob_BB, "./Checkpoint/nsbomi4_3_checkpoint.mpl":
      fi;

    od:

    if success_count = 0 then
      mean_num_bursts_corrected := 0;
    else
      mean_num_bursts_corrected:=total_num_bursts_corrected/success_count;
    fi;

    avg_num_bursts := total_num_bursts/runs;

    if total_num_bursts = 0 then
      mean_burst_length := 0;
    else
      mean_burst_length := error_count/total_num_bursts;
    fi;
```

```
######################### Begin Tables ###################################


   ### Code to make the Experiment Parameters table
   ### Note this table is only made when all boolean values are true
   ### This is because the values in this table are the same for all
   ### possible boolean value.


    if Stack=true and Burst_Only=true and minimal_interleave=true and
    P_GG = 0.9999 then

      exp_par_list[1] := rho;
      exp_par_list[2] := error_count/(depth*L*runs);
      exp_par_list[3] := bl;
      exp_par_list[4] := mean_burst_length;
      exp_par_list[5] := rho*depth*L/bl;
      exp_par_list[6] := avg_num_bursts;

      exp_header_list[1] := d_f;
      exp_header_list[2] := c;
      exp_header_list[3] := c_hat;
      exp_header_list[4] := Ebrst;
      exp_header_list[5] := NonEbrst;
      exp_header_list[6] := ml;
      exp_header_list[7] := success_count/runs*100;
      exp_header_list[8] := total_pseudo_sols/runs*100;
      exp_header_list[9] := max_num_bursts_corrected;
      exp_header_list[10] := mean_num_bursts_corrected;
      exp_header_list[11] := max_burst_length_corrected;
      exp_header_list[12] := total_num_bursts;
      exp_header_list[13] := total_num_bursts_corrected;
      exp_header_list[14] := max_burst_length;
      exp_header_list[15] := max_num_bursts;


      if P_BB = 0.9 then
        row := 1;
      elif P_BB = 0.85 then
        row := 2;
      else
        row := 3;
      fi;

      for i from 1 to nops(exp_par_list) do
        exp_par_tbl_mat[row, i] := exp_par_list[i];
      od:

      for i from 1 to nops(exp_header_list) do
        sta_bol_min_tbl_mat[row, i] := exp_header_list[i];
      od:
      save row, exp_par_tbl_mat, cat(path_maple_files,
           "exp_par_tbl_3rows.mpl");
      save row, sta_bol_min_tbl_mat, cat(path_maple_files,
      "sta_bol_min_tbl_3rows.mpl");
      Make_Tbls:-Make_Exp_Par_Tbl();
      tbl_id := "sbomi";
      Make_Tbls:-Make_Exp_Tbls(tbl_id);
      FileTools[Remove]("./Checkpoint/sbomi1_1_checkpoint.mpl");
      FileTools[Copy]("./Checkpoint/init_checkpoint.mpl",
      "./Checkpoint/sbomi1_1_checkpoint.mpl");

    fi;  ### end outer if block

    if Stack=true and Burst_Only=true and minimal_interleave=true and
    P_GG = 0.999 then

      exp_par_list[1] := rho;
      exp_par_list[2] := error_count/(depth*L*runs);
      exp_par_list[3] := bl;
      exp_par_list[4] := mean_burst_length;
      exp_par_list[5] := rho*depth*L/bl;
      exp_par_list[6] := avg_num_bursts;
```

```
exp_header_list[1] := d_f;
exp_header_list[2] := c;
exp_header_list[3] := c_hat;
exp_header_list[4] := Ebrst;
exp_header_list[5] := NonEbrst;
exp_header_list[6] := ml;
exp_header_list[7] := success_count/runs*100;
exp_header_list[8] := total_pseudo_sols/runs*100;
exp_header_list[9] := max_num_bursts_corrected;
exp_header_list[10] := mean_num_bursts_corrected;
exp_header_list[11] := max_burst_length_corrected;
exp_header_list[12] := total_num_bursts;
exp_header_list[13] := total_num_bursts_corrected;
exp_header_list[14] := max_burst_length;
exp_header_list[15] := max_num_bursts;



if P_BB = 0.9 then
  row := 4;
elif P_BB = 0.85 then
  row := 5;
else
  row := 6;
fi;

for i from 1 to nops(exp_par_list) do
  exp_par_tbl_mat[row, i] := exp_par_list[i];
od:

for i from 1 to nops(exp_header_list) do
  sta_bol_min_tbl_mat[row, i] := exp_header_list[i];
od:
save row, exp_par_tbl_mat, cat(path_maple_files,
     "exp_par_tbl_3rows.mpl");
save row, sta_bol_min_tbl_mat, cat(path_maple_files,
"sta_bol_min_tbl_3rows.mpl");
Make_Tbls:-Make_Exp_Par_Tbl();
```

```
    tbl_id := "sbomi";
    Make_Tbls:-Make_Exp_Tbls(tbl_id);
    FileTools[Remove]("./Checkpoint/sbomi1_2_checkpoint.mpl");
    FileTools[Copy]("./Checkpoint/init_checkpoint.mpl",
    "./Checkpoint/sbomi1_2_checkpoint.mpl");

  fi;  ### end outer if block

  if Stack=true and Burst_Only=true and minimal_interleave=true and
  P_GG = 0.99 then

    exp_par_list[1] := rho;
    exp_par_list[2] := error_count/(depth*L*runs);
    exp_par_list[3] := bl;
    exp_par_list[4] := mean_burst_length;
    exp_par_list[5] := rho*depth*L/bl;
    exp_par_list[6] := avg_num_bursts;

    exp_header_list[1] := d_f;
    exp_header_list[2] := c;
    exp_header_list[3] := c_hat;
    exp_header_list[4] := Ebrst;
    exp_header_list[5] := NonEbrst;
    exp_header_list[6] := ml;
    exp_header_list[7] := success_count/runs*100;
    exp_header_list[8] := total_pseudo_sols/runs*100;
    exp_header_list[9] := max_num_bursts_corrected;
    exp_header_list[10] := mean_num_bursts_corrected;
    exp_header_list[11] := max_burst_length_corrected;
    exp_header_list[12] := total_num_bursts;
    exp_header_list[13] := total_num_bursts_corrected;
    exp_header_list[14] := max_burst_length;
    exp_header_list[15] := max_num_bursts;

    if P_BB = 0.9 then
      row := 7;
    elif P_BB = 0.85 then
      row := 8;
    else
      row := 9;
    fi;

    for i from 1 to nops(exp_par_list) do
      exp_par_tbl_mat[row, i] := exp_par_list[i];
    od:

    for i from 1 to nops(exp_header_list) do
      sta_bol_min_tbl_mat[row, i] := exp_header_list[i];
    od:

    save row, exp_par_tbl_mat, cat(path_maple_files,
         "exp_par_tbl_3rows.mpl");
    save row, sta_bol_min_tbl_mat, cat(path_maple_files,
    "sta_bol_min_tbl_3rows.mpl");
    Make_Tbls:-Make_Exp_Par_Tbl();
    tbl_id := "sbomi";
    Make_Tbls:-Make_Exp_Tbls(tbl_id);
    FileTools[Remove]("./Checkpoint/sbomi1_3_checkpoint.mpl");
    FileTools[Copy]("./Checkpoint/init_checkpoint.mpl",
    "./Checkpoint/sbomi1_3_checkpoint.mpl");

  fi;  ### end outer if block

  if Stack=false and Burst_Only=true and minimal_interleave=true and
  P_GG = 0.9999 then

    exp_header_list[1] := d_f;
    exp_header_list[2] := c;
    exp_header_list[3] := c_hat;
    exp_header_list[4] := Ebrst;
    exp_header_list[5] := NonEbrst;
    exp_header_list[6] := ml;
    exp_header_list[7] := success_count/runs*100;
    exp_header_list[8] := total_pseudo_sols/runs*100;
    exp_header_list[9] := max_num_bursts_corrected;
    exp_header_list[10] := mean_num_bursts_corrected;
    exp_header_list[11] := max_burst_length_corrected;
    exp_header_list[12] := total_num_bursts;
    exp_header_list[13] := total_num_bursts_corrected;
    exp_header_list[14] := max_burst_length;
    exp_header_list[15] := max_num_bursts;

    if P_BB = 0.9 then
      row := 1;
    elif P_BB = 0.85 then
      row := 2;
    else
      row := 3;
    fi;

    for i from 1 to nops(exp_header_list) do
      not_sta_bol_min_tbl_mat[row, i] := exp_header_list[i];
    od:

    save row, not_sta_bol_min_tbl_mat, cat(path_maple_files,
    "not_sta_bol_min_tbl_3rows.mpl");
    tbl_id := "nsbomi";
    Make_Tbls:-Make_Exp_Tbls(tbl_id);
    FileTools[Remove]("./Checkpoint/nsbomi4_1_checkpoint.mpl");
    FileTools[Copy]("./Checkpoint/init_checkpoint.mpl",
    "./Checkpoint/nsbomi4_1_checkpoint.mpl");

  fi;  ### end outer if block

  if Stack=false and Burst_Only=true and minimal_interleave=true and
  P_GG = 0.999 then

    exp_header_list[1] := d_f;
    exp_header_list[2] := c;
    exp_header_list[3] := c_hat;
    exp_header_list[4] := Ebrst;
```

```
    exp_header_list[5] := NonEbrst;
    exp_header_list[6] := ml;
    exp_header_list[7] := success_count/runs*100;
    exp_header_list[8] := total_pseudo_sols/runs*100;
    exp_header_list[9] := max_num_bursts_corrected;
    exp_header_list[10] := mean_num_bursts_corrected;
    exp_header_list[11] := max_burst_length_corrected;
    exp_header_list[12] := total_num_bursts;
    exp_header_list[13] := total_num_bursts_corrected;
    exp_header_list[14] := max_burst_length;
    exp_header_list[15] := max_num_bursts;

    if P_BB = 0.9 then
      row := 4;
    elif P_BB = 0.85 then
      row := 5;
    else
      row := 6;
    fi;

    for i from 1 to nops(exp_header_list) do
      not_sta_bol_min_tbl_mat[row, i] := exp_header_list[i];
    od:
    save row, not_sta_bol_min_tbl_mat, cat(path_maple_files,
    "not_sta_bol_min_tbl_3rows.mpl");
    tbl_id := "nsbomi";
    Make_Tbls:-Make_Exp_Tbls(tbl_id);
    FileTools[Remove]("./Checkpoint/nsbomi4_2_checkpoint.mpl");
    FileTools[Copy]("./Checkpoint/init_checkpoint.mpl",
    "./Checkpoint/nsbomi4_2_checkpoint.mpl");

  fi;  ### end outer if block

  if Stack=false and Burst_Only=true and minimal_interleave=true and
  P_GG = 0.99 then

    exp_header_list[1] := d_f;
    exp_header_list[2] := c;
    exp_header_list[3] := c_hat;
    exp_header_list[4] := Ebrst;
    exp_header_list[5] := NonEbrst;
    exp_header_list[6] := ml;
    exp_header_list[7] := success_count/runs*100;
    exp_header_list[8] := total_pseudo_sols/runs*100;
    exp_header_list[9] := max_num_bursts_corrected;
    exp_header_list[10] := mean_num_bursts_corrected;
    exp_header_list[11] := max_burst_length_corrected;
    exp_header_list[12] := total_num_bursts;
    exp_header_list[13] := total_num_bursts_corrected;
    exp_header_list[14] := max_burst_length;
    exp_header_list[15] := max_num_bursts;

    if P_BB = 0.9 then
      row := 7;
    elif P_BB = 0.85 then
      row := 8;
    else
      row := 9;
    fi;

    for i from 1 to nops(exp_header_list) do
      not_sta_bol_min_tbl_mat[row, i] := exp_header_list[i];
    od:

    save row, not_sta_bol_min_tbl_mat, cat(path_maple_files,
    "not_sta_bol_min_tbl_3rows.mpl");
    tbl_id := "nsbomi";
    Make_Tbls:-Make_Exp_Tbls(tbl_id);
    FileTools[Remove]("./Checkpoint/nsbomi4_3_checkpoint.mpl");
    FileTools[Copy]("./Checkpoint/init_checkpoint.mpl",
    "./Checkpoint/nsbomi4_3_checkpoint.mpl");

  fi;  ### end outer if block
```

```
##################### End of Tables #################################

  end proc; ### End of Pol_Vec_Rec_Slv

  KW_Interleave_Parameters :=
  proc({P_GG :: float := 0.00 ### The prob. of staying in the good state
        ,P_BB :: float := 0.00 ### The prob. of staying in the bad state
        ,ML   :: integer := 0 ### The length of the message
        ,Stack:: boolean := true ### the interleaves stacked
        ,Burst_Only :: boolean := true ### only burst error considered
        ,minimal_interleave :: boolean := true ### inteleave depth is 3
        })
    local Ebrst ### The upper bound for the number of burst errors
         ,NonEbrst ### The upper bound for the number of non burst errors
         ,d_f ### The degree of the polynomilas
         ,rho ### The symbol error rate
         ,bl ### The average burst length
         ,ml ### The message length
         ,c_hat ### The depth of the interleave
         ,c ### One less the number of polynomials in the interleave
         ,wrap_around
         ,i ### index
         ;

    rho := (1 - P_GG)/(1 - P_BB + 1 - P_GG); ## the symbol error rate

    bl := ceil(1/(1 - P_BB)); ### Formula for the average burst length

    ml := ML; ### Retrieve the message length input by the user

    if minimal_interleave then

      c_hat := 3; ### 3 is the shallowest interleave depth with wrap around

    else

      c_hat := bl; ### The depth of the interleave, or average burst length
```

```
fi;

c := c_hat;

Ebrst := 2*ceil(rho*ml/bl);

if Burst_Only then

  NonEbrst := 0;

else

  NonEbrst := Ebrst;

fi;

d_f := (2*Ebrst+2*NonEbrst+ )*c_hat- (Ebrst+2*NonEbrst + 1)*c -
       (2*Ebrst+2*NonEbrst+1);

for i from 1 to Ebrst while d_f < 0 do
  NonEbrst := NonEbrst - 1;
  d_f := (2*Ebrst+2*NonEbrst+1)*c_hat-(Ebrst+2*NonEbrst+1)*c -
         (2*Ebrst+2*NonEbrst+1);
od:


if d_f < 0 then

  error "The polynomial degree is negative";

fi;

for c from c_hat  while ((d_f + 2*Ebrst + 2*NonEbrst + 1) +
c*(d_f + Ebrst + 2*NonEbrst + 1) <>
c_hat*(d_f + 2*Ebrst + 2*NonEbrst + 1)) do ### wrap around check
### we can increase c if there is no wrap around
```

```
      d_f := (2*Ebrst+2*NonEbrst+1)*c_hat-(Ebrst+2*NonEbrst+1)*(c+1) -
      (2*Ebrst+2*NonEbrst+1)/(c+1-c_hat+1);

      if d_f < 0 then

        error "The polynomial degree is negative";

      fi;

      if not df::integer then

        next;

      fi;

    od:

    message:-Interleaver(d_f, ml, Ebrst, NonEbrst, c_hat, c);

    return Pol_Vec_Rec_Slv(Ebrst, NonEbrst, d_f, P_GG, P_BB, c_hat, c,
           Stack, Burst_Only, minimal_interleave, rho, bl, ml);

  end proc; ### End of KW_Non_Stack_Interleave_Parameters

end module:
```

End of code

Start of code

```
### This file contains the polynomial vector along with the interleaver

message := module()

  export Interleaver   ### The function that handles the interleaving
         ,phi_interleave ### Interleave of the phi_vector
```

```
       ,pol_interleave ### Interleave of the pol_vector
       ,pol_vector   ### The message encoded as polynomials.
      ;

 Interleaver := proc(d_f, ml, Ebrst, NonEbrst, c_hat, c)

 local ### pol_vector   ### The message encoded as polynomials.
        phi_vector ### vector of polynomials in the coefficient phi
       ,i ### used as an index
       ,j ### used as an index
       ,k ### used as an index
       ,num_pol ### Number of polynomials required for the message
       ,interleave_depth ### The depth of the stacks
       ,phi_list
       ,pol_list
       ,LBK
       ,counter
       ;

pol_list := [];
phi_list := [];

print(d_f);

num_pol := ml/(d_f + 1); ### the num of polynomials in the message

phi_vector := Vector(num_pol, i->sum(phi[i][j]*u^j,
                j=0..d_f+Ebrst+NonEbrst));

pol_vector := Vector(num_pol);

for i from 1 to num_pol do
  randomize(ithprime(i));
  pol_vector[i] := randpoly([u], degree = d_f);
od:

LBK := d_f + 2*(Ebrst + NonEbrst) + 1;
```

```
interleave_depth := c_hat*(ml/((c+1)*(d_f + 1)));

phi_interleave := Matrix(interleave_depth, LBK);

pol_interleave := Matrix(interleave_depth, LBK);

counter := 0;
k := 0;
for i from 1 to num_pol do
  if i mod (c+1) = 1 then
    for j from 1 to LBK do
      pol_list := [op(pol_list), pol_vector[i]];
      phi_list := [op(phi_list), phi_vector[i]];
    od:
  else
    for j from 1 to LBK-Ebrst do
      pol_list := [op(pol_list), pol_vector[i]];
      phi_list := [op(phi_list), phi_vector[i]];
    od:
  fi;
od:
k := 1;
for i from 1 to interleave_depth do
  for j from 1 to LBK do
    pol_interleave[i, j] := pol_list[k];
    phi_interleave[i, j] := phi_list[k];
    k := k + 1;
  od:
od:

for i from 1 to interleave_depth - (c_hat - 1) by c_hat do
### center complete Reed-Solomon
  LinearAlgebra:-RowOperation(pol_interleave, [i, i+1], inplace = true);
  LinearAlgebra:-RowOperation(phi_interleave, [i, i+1], inplace = true);
od:
  end proc; ### End of Interleaver

end module:
```

End of code

Start of Code

```
### This file contains a module that makes the newcommands for the
### experiment tables in the dissertation

Make_Tbls := module()

  export Make_Exp_Par_Tbl
        ,Make_Exp_Tbls
        ;

  local num ### the list one, two, three, etc
        ;

  num := [seq(i, i = 1 .. 9)];
  num[1] := "One";
  num[2] := "Two";
  num[3] := "Three";
  num[4] := "Four";
  num[5] := "Five";
  num[6] := "Six";
  num[7] := "Seven";
  num[8] := "Eight";
  num[9] := "Nine";

  Make_Exp_Par_Tbl := proc() ### make Experiment Parameters Table

    local col_name_list  ### names of the columns in the table
         ,i ### index
```

```
       ,j ### index
       ;

  read("../Runs/Maple_Files/exp_par_tbl_3rows.mpl");
  col_name_list := [seq(i, i = 1 .. 6)];
  col_name_list[1] := "AnaSymErrRate";
  col_name_list[2] := "ActSymErrRate";
  col_name_list[3] := "AnaMeanBurstLen";
  col_name_list[4] := "ActMeanBurstLen";
  col_name_list[5] := "AnaMeanNumBurst";
  col_name_list[6] := "ActMeanNumBurst";

  fopen("../Runs/Latex_Files/exp_par_tbl_3rows.tex", APPEND);

  for j from 1 to 6 do
    if j = 1 or j = 2 then
    fprintf("../Runs/Latex_Files/exp_par_tbl_3rows.tex",
              "%s%s%s%s%f%s\n\n", "\\newcommand{\\",col_name_list[j],
              num[row],"}{", exp_par_tbl_mat[row][ j], "}");
    else

    fprintf("../Runs/Latex_Files/exp_par_tbl_3rows.tex",
              "%s%s%s%s%0.2f%s\n\n", "\\newcommand{\\",col_name_list[j],
              num[row],"}{", exp_par_tbl_mat[row][ j], "}");
    fi;
  od:
  fclose("../Runs/Latex_Files/exp_par_tbl_3rows.tex");

end proc; ### End of make the Experiment Parameters Table

Make_Exp_Tbls := proc(tbl_id)
  local col_name_list  ### names of the columns in the table
       ,i ### index
       ,j ### index
       ;

  col_name_list := [seq(i, i = 1 .. 15)];
```

```
col_name_list[1] := "PolDeg";
col_name_list[2] := "c";
col_name_list[3] := "cHat";
col_name_list[4] := "BrstErr";
col_name_list[5] := "NonBrstErr";
col_name_list[6] := "MessLen";
col_name_list[7] := "PctCorMes";
col_name_list[8] := "PctPsudoVecCom";
col_name_list[9] := "MaxNumBurstCor";
col_name_list[10] := "MeanNumBurstCor";
col_name_list[11] := "MaxBurstLenCor";
col_name_list[12] := "TotNumBursts";
col_name_list[13] := "TotNumBurstsCorrected";
col_name_list[14] := "MaxBurstLength";
col_name_list[15] := "MaxNumBursts";

if tbl_id = "sbomi" then
  read("../Runs/Maple_Files/sta_bol_min_tbl_3rows.mpl");
  if row = 1 then
    fopen("../Runs/Latex_Files/sta_bol_min_tbl_3rows.tex", WRITE);
  else
    fopen("../Runs/Latex_Files/sta_bol_min_tbl_3rows.tex", APPEND);
  fi;

  for j from 1 to LinearAlgebra:-ColumnDimension(sta_bol_min_tbl_mat) do
    if j < 7 then
      fprintf("../Runs/Latex_Files/sta_bol_min_tbl_3rows.tex",
      "%s%s%s%s%s%d%s\n\n", "\\newcommand{\\", tbl_id, col_name_list[j],
      num[row],"}{", sta_bol_min_tbl_mat[row][j], "}");
    else
      fprintf("../Runs/Latex_Files/sta_bol_min_tbl_3rows.tex",
      "%s%s%s%s%s%0.2f%s\n\n", "\\newcommand{\\", tbl_id,
      col_name_list[j], num[row],"}{", sta_bol_min_tbl_mat[row][j],
      "}");
      fi;
  od:
  fclose("../Runs/Latex_Files/sta_bol_min_tbl_3rows.tex");
```

```
    else tbl_id = "nsbomi" then
      read("../Runs/Maple_Files/not_sta_bol_min_tbl_3rows.mpl");
      fopen("../Runs/Latex_Files/not_sta_bol_min_tbl_3rows.tex", APPEND);
      for j from 1 to LinearAlgebra:-ColumnDimension
      (not_sta_bol_min_tbl_mat) do
        if j < 7 then
          fprintf("../Runs/Latex_Files/not_sta_bol_min_tbl_3rows.tex",
          "%s%s%s%s%s%d%s\n\n", "\\newcommand{\\", tbl_id, col_name_list[j],
          num[row],"}{", not_sta_bol_min_tbl_mat[row][j], "}");
        else
          fprintf("../Runs/Latex_Files/not_sta_bol_min_tbl_3rows.tex",
          "%s%s%s%s%s%0.2f%s\n\n", "\\newcommand{\\", tbl_id,
          col_name_list[j], num[row],"}{", not_sta_bol_min_tbl_mat[row][j],
          "}");
        fi;
      od:
      fclose("../Runs/Latex_Files/not_sta_bol_min_tbl_3rows.tex");

    fi;

  end proc;

end module:
```

End of code