7-27-2020 10:00 AM

Hybrid Symbolic-Numeric Computing in Linear and Polynomial Algebra

Leili Rafiee Sevyeri, *The University of Western Ontario*

Supervisor: Robert M. Corless, *The University of Western Ontario*

A thesis submitted in partial fulfillment of the requirements for the Doctor of Philosophy degree in Applied Mathematics

Abstract

In this thesis, we introduce hybrid symbolic-numeric methods for solving problems in linear and polynomial algebra. We mainly address the *approximate* `GCD` problem for polynomials, and problems related to *parametric* and *polynomial matrices*. For symbolic methods, our main concern is their complexity and for the numerical methods we are more concerned about their stability. The thesis consists of 5 articles which are presented in the following order:

Chapter 1, deals with the fundamental notions of conditioning and backward error. Although our results are not novel, this chapter is a novel explication of conditioning and backward error that underpins the rest of the thesis.

In Chapter 2, we adapt Victor Y. Pan's root-based algorithm for finding approximate `GCD` to the case where the polynomials are expressed in Bernstein bases. We use the numerically stable companion pencil of Guðbjörn Jónsson to compute the roots, and the Hopcroft-Karp bipartite matching method to find the degree of the approximate `GCD`. We offer some refinements to improve the process.

In Chapter 3, we give an algorithm with similar idea to Chapter 2, which finds an approximate `GCD` for a pair of approximate polynomials given in a Lagrange basis. More precisely, we suppose that these polynomials are given by their approximate values at distinct known points. We first find each of their roots by using a Lagrange basis companion matrix for each polynomial. We introduce new clustering algorithms and use them to cluster the roots of each polynomial to identify multiple roots, and then *marry* the two polynomials using a *Maximum Weight Matching (MWM)* algorithm, to find their `GCD`.

In Chapter 4, we define "generalized standard triples" $\boldsymbol{X}$, $z\boldsymbol{C}_1 - \boldsymbol{C}_0$, $\boldsymbol{Y}$ of regular matrix polynomials $\boldsymbol{P}(z) \in \mathbb{C}^{n\times n}$ in order to use the representation $\boldsymbol{X}(z\boldsymbol{C}_1 - \boldsymbol{C}_0)^{-1}\boldsymbol{Y} = \boldsymbol{P}^{-1}(z)$ for $z \notin \Lambda(\boldsymbol{P}(z))$. This representation can be used in constructing algebraic linearizations; for example, for $\boldsymbol{H}(z) = z\boldsymbol{A}(z)\boldsymbol{B}(z) + \boldsymbol{C} \in \mathbb{C}^{n\times n}$ from linearizations for $\boldsymbol{A}(z)$ and $\boldsymbol{B}(z)$. This can be done even if $\boldsymbol{A}(z)$ and $\boldsymbol{B}(z)$ are expressed in differing polynomial bases. Our main theorem is that $\boldsymbol{X}$ can be expressed using the coefficients of the expression $1 = \sum_{k=0}^{\ell} e_k\phi_k(z)$ in terms of the relevant polynomial basis. For convenience we tabulate generalized standard triples for orthogonal polynomial bases, the monomial basis, and Newton interpolational bases; for the Bernstein basis; for Lagrange interpolational bases; and for Hermite interpolational bases. We account for the possibility of common similarity transformations. We give explicit proofs for the less familiar bases.

Chapter 5 is devoted to parametric linear systems (PLS) and related problems, from a symbolic computational point of view. PLS are linear systems of equations in which some symbolic parameters, that is, symbols that are not considered to be candidates for elimination or solution in the course of analyzing the problem, appear in the coefficients of the system. We assume that the symbolic parameters appear polynomially in the coefficients and that the only variables to be solved for are those of the linear system. It is well-known that it is possible to specify a covering set of regimes, each of which is a semi-algebraic condition on the parameters together with a solution description valid under that condition. We provide a method of solution that requires time polynomial in the matrix dimension and the degrees of the polynomials when there are up to three parameters. Our approach exploits the Hermite and Smith normal forms that may be computed when the system coefficient domain is mapped to the univariate polynomial domain over suitably constructed fields. Our approach effectively identifies *intrinsic singularities* and *ramification points* where the algebraic and geometric structure of the matrix changes. Specially parametric eigenvalue problems can be addressed as well. Although we do not directly address the problem of computing the Jordan form, our

approach allows the construction of the algebraic and geometric eigenvalue multiplicities revealed by the Frobenius form, which is a key step in the construction of the Jordan form of a matrix.

Keywords: Interpolation, Rootfinding, Conditioning, Sensitivity, Bernstein basis, Approximate `GCD` Maximum matching, Bipartite graph, Root clustering, Companion pencil, Lagrange basis, Maximum weight matching, Hermite form, Smith form, Frobenius form, Parametric linear systems, Standard triple, Regular matrix polynomial, Polynomial bases, Companion matrix, Colleague matrix, Comrade matrix, Algebraic linearization, Linearization of matrix polynomials.

Summary for Lay Audience

Matrices as arrays of numbers and polynomials as the simplest type of mathematical functions naturally arise in a majority of computational problems. Such useful objects are studied well in the literature due to their vast applications in science and engineering. Solving linear systems of equations and finding roots of polynomials are probably the most well understood problems. However, efforts to introduce more efficient and/or reliable algorithms in general or for special cases are going on.

In this thesis, we focus on a few problems related to matrices and polynomials over complex numbers with a hybrid symbolic-numeric approach. We present algorithms for approximate Greatest Common Divisor (`GCD`) of polynomials in Bernstein and Lagrange bases. Another problem we have addressed is solving linear systems with parameters as coefficients. Relevant problems to linearization of matrix polynomials in multiple bases are discussed as well.

Co-Authorship

This integrated-article thesis is based on 5 papers. Chapter 1 [3] and Chapter 2 [1] have been published in academic journals. Chapter 3 is being prepared for publication. Chapters 4 [2] and 5 have been submitted for publication. In all these papers, Robert Corless contributed in producing the material and he also helped with the theoretical understanding and feedback. [2] is a joint work with Eunice Chan and Rob Corless. Chapter 5 is a joint work with Rob Corless, Mark Giesbrecht and B. David Saunders.

Bibliography

[1] R. M. Corless and L. Rafiee Sevyeri. Approximate GCD in a Bernstein basis. *Springer International Publishing*, 77–91, 2019.

[2] E. Y. S. Chan, R. M. Corless and L. Rafiee Sevyeri. Generalized Standard Triples for Algebraic Linearizations of Matrix Polynomials. *arXiv 1805.04488*, 2018.

[3] R. M. Corless and L. Rafiee Sevyeri. The Runge Example for Interpolation and Wilkinson's Examples for Rootfinding. *SIAM Review*, 62(1) 231-243, 2018.

To Armin and our little girl

Acknowledgements

I am surrounded with a lot of good people that I need to thank for. Without their help, this thesis may not have been possible.

First of all, I would like to thank my supervisor, Rob Corless. He is the most caring, supportive and kind supervisor that someone could wish for. He supported me all these years through rough times. He taught me how to be a better researcher and mostly how to become a better person. I am very grateful to have him as my mentor. Thank you Rob.

I would like to thank the lab members of Ontario Research Centre for Computer Algebra (ORCCA) for all these years, specially, I have to thank David Jeffrey and Greg Reid for their constant positive attitude and for all the encouragements they gave throughout the process.

Special thanks goes to Mark Giesbrecht and George Labahn, for supporting me while I was a visiting researcher in the Symbolic Computation Group (SCG) in department of computer science at the University of Waterloo in the last two years of my PhD studies. I also want to thank my friends in SCG for their kindness and support.

Huge thanks goes to David Saunders, who I had the opportunely to work with in the last year of my PhD. I learned a lot from him and he was extremely supportive.

I am proud, honored and extremely grateful to have been able to work with such amazingly talented researchers, who are also extremely kind human beings, Rob, David, Mark, George and Dave.

I also want to thank our beloved academic program coordinator in the department of applied mathematics, Audrey, for her kindness and positive energy through all these years.

I also thank the examination board, Mark Daley, Christopher Essex, David Jeffrey and Laureano Gonzalez Vega, for their valuable feedback and comments.

After all these people who helped me with my academic journey, I want to thank my non-academic family and friend.

First, I believe I need to thank my friend, my partner and my love, Armin for his unconditional support through all these years. I am very grateful to have you beside me. I could not achieve what I have achieved so far without your help. You helped me in many ways. You taught me to be strong, to be a better person and better researcher. I am very grateful for having you in my life. You are the only reason that I am standing here. Everything that I have is because of you and your true love and support.

I am grateful for my dearest friends, Fati and Haleh for their love and support. Even though you are thousands miles away from me, you have always been there for me whenever I needed your positive energy and love. I also want to thank my lovely angel, Farnaz.

I am also thankful to my in-laws for their unconditional support and love through all these years.

Last but not least, I want to thank my parents. Maman and Baba, I am so thankful to you for not giving up on me, even for one moment. I admire you for the selfless love, care, pain and sacrifice you did for me. Words can not describe my feelings to you and show how grateful I am to be your daughter. You raised me in a way that I never give up on my dreams and always follow my heart. You supported me no matter what other say and I really appreciate this. I also wish to thank my sisters, Laya and Sara for their love and kindness. You are more than sisters to me. I could not wish for better sisters than you. Thanks for being there for me whenever I needed you the most, without even asking.

Contents

List of Figures

List of Tables

Chapter 0

Introduction

This thesis introduces hybrid symbolic-numeric methods which solve problems related to polynomials, matrices and matrix polynomials. These objects are of huge interest due to their vast applications in engineering and science. A large part of the literature in computational mathematics is devoted to studying matrices and polynomials, and even today a lot of effort is going on to extend our knowledge about them.

This short introduction is an effort to briefly present the essence of hybrid symbolic-numeric computational methods, and is in no way comprehensive or detailed. We invite the reader to consult [4] to get a better understanding of hybrid symbolic-numeric computation. Since each individual chapter of this thesis comes with its own introduction, here we only give an overview and the general relations between these chapters.

The exact dates for the invention or discovery of computational mathematics is not known, but records from Babylonian/Sumerian times, such as clay tablets (date from 1800 to 1600 BC), are known [3]. Although the terms "symbolic" and "numeric" are fairly new, the methods have been used since ancient times.

Babylonians knew 1.414222 is an approximation of $\sqrt{2}$. It is also known that Babylonian used to compute the area of a circle by computing 3 times the square of the radius. We also have evidence which shows that Babylonian had an approximation of π as 3.125 (it was not presented in decimals. For more detail, see [3, P. 35]). There are many more examples of approximations from ancient mathematics. Rather than giving a list of such computational methods, our point is that, non-exact computation is being used since ancient times. In recent years all these methods are considered as part of *numerical* methods.

The oldest well-known computational method which is still in use is probably the Euclidean algorithm [2]. The method was first introduced by Euclid in his treatise, *Elements*. Indeed at the time no one called it the Euclidean algorithm, but the method was being used since then. Surprisingly, this method is still being used extensively for finding *the greatest common divisor* (`GCD`) of two integers (or more generally of two polynomials with coefficients in over a field). Such computation which is exact in the sense of both result and the intermediate steps, are considered as a part of *symbolic* computation these days. Symbolic computation is the study of mathematical objects and structures containing symbols. One should note that such symbols can be of different type. For example, in an equation, variables and parameters are symbols. While variables are the ones which are typically solved for, parameters are typically carried in the computation and may appear in the solutions. See Chapter 5 for more detail.

By development of mathematics, objects and structures became more formally understood. Eventually the two mentioned computational approaches, exact and approximate, became more formal with their own tools and subjects of interests. In the recent decades a lot of efforts have been made to develop each of these independently into a modern language. Today we know these approaches (respectively) as symbolic and numeric computation.

There are mathematical objects of which both symbolic and numeric computational methods are applicable to. However, there are areas such as finite fields computation which can not be approached with numerical computation in an straightforward way. Although both types of methods are available for specific type of computations, goals are different for numeric and symbolic computations. Here we try to present some properties of both approaches.

Assume we need to work with $\sqrt{2}$, the second root of 2. In symbolic computation $\sqrt{2}$ is considered as a number which is a root of polynomial x^2-2 and will be treated in computations as a symbol with its own properties. In the numerical approach, we do not work with such symbol, rather than that we work with its approximation $\sqrt{2}\approx 1.4142$. Note that the number of decimal points considered is up to us and we may consider more digits.

The difference between symbolic and numeric computations is not limited to how they treat numbers (objects). The main concern in symbolic computation is complexity (cost) of an algorithm. The complexity is also important in numeric computation, but it is not the first priority. The main concern in numeric computation is error. These errors may naturally arise in the input or in the process of our algorithm. There are two major notions in numeric computation which are the first priorities and are relevant to errors. We briefly talk about *condition* of a problem and *stability* of an algorithm here and we invite the reader to consult [1] for more details.

Instead of formal definitions of condition number and stability of an algorithm, we present examples which gives the ideas. Assume $f(x)=(x-1)(x-2)=x^2-3x+2$ and $g(x)=x-2$. It is clear that $\gcd(f(x),g(x))=x-2$. Now assume we have a small perturbation in the constant term of g and we get $g(x)=x-2.0000001$. The error in the input causes that $\gcd(f(x),g(x))=1$. Hence a small error in input leads to a trivial `GCD` while we expect a nontrivial one. Note that the trivial `GCD` has nothing to do with the algorithm we use for finding the `GCD` . This is a property of the problem, which is finding the `GCD` . Condition of a problem measures how the result is changed for small errors in the input. In case that the error in the result is big, similar to what we detected in the `GCD` problem, we call the problem *ill-conditioned* (in fact, the `GCD` problem is even *ill-posed*, that is, it fails to be a *well-posed* problem, which must have a unique solution which depends continuously on the input. Here the `GCD` is discontinuous in the input).

Now consider the polynomial $f(x)=ax^2+bx+c$. The roots of f are given by

$$x_1=\frac{-b+\sqrt{b^2-4ac}}{2a},\quad x_2=\frac{-b-\sqrt{b^2-4ac}}{2a} \tag{0.0.1}$$

In case that $b\gg 4ac$, $|b|\approx\sqrt{b^2-4ac}$. This may lead to a huge relative error for computing x_1(or x_2 depending on b), which is caused by round off errors. However, these errors are caused by our method for finding the roots. We could avoid it by changing our method and use the fact that $x_1x_2=\frac{c}{a}$. This way we can avoid subtraction of very close numbers. The former algorithm for finding the roots of f (using formulas in (0.0.1)) is called *unstable* since it produces large errors in the result. A careful definition of stability depends on meaning of "large error".

Both of the above notions are very important in numerical methods. While designing stable algorithms is the responsibility of numerical analysts, modelers are in charge

of making well-conditioned problems. However, that is just in theory, in practice a numerical analyst usually tries to replace ill-posed problems with a similar problem which is well-posed. One can name the *approximate* `GCD` problem as such effort which is studied in Chapters 2 and 3.

In general the main concern in numerical computation is stability and condition number. While we also care about complexity, our first priority is to avoid producing large errors in our computations.

Although symbolic and numeric approaches are in general considered independent, one can combine methods from one with another. This mixed approach is getting more popular these days and is called *hybrid computation*. The development of such methods are due to the demands of efficiency and reliability in scientific computing.

As it was mentioned at the beginning, in this thesis we are concerned with polynomials, matrices, and polynomial matrices which are isomorphic to matrix polynomials. These play a fundamental role in computation for modeling and design and control.

Particularly important topics are

- rootfinding: for design, one wants to have a value of an input that produces a desired output. This can be phrased as a rootfinding problem, $p(x) - y = 0$, where y is the desired value of the polynomial $p(x)$. There are many other applications of rootfinding which we do not state here.

- eigenvalues: many problems can be modeled using linear transformations which can be written as matrices. Informally, an eigenvalue λ is a scalar such that, a matrix $\boldsymbol{A}$ acting on a vector $\vec{v}$ is the same as the scalar λ acting on $\vec{v}$. Geometrically, λ can be seen as an stretching factor for the action of $\boldsymbol{A}$ on specific vectors.

- polynomial eigenvalues: for $P(x)$ a matrix polynomial, find λ such that $P(\lambda)$ is singular, which is equivalent to finding λ and $\vec{v}$ such that $P(\lambda)\vec{v}$ is the zero vector.

Rootfinding and eigenvalues are connected by companion matrices, companion pencils, and linearizations (see Chapter 5 for more details).

Given the multi-thousand year history of computational mathematics, it is surprising that there are new things to be said. The new things in this thesis appear in Chapters 2, 3, 4, and 5.

Chapter 1 deals with the fundamental notions of conditioning and backward error (and a little bit with complexity). Chapter 1 is not novel, per se, but is a novel explication of conditioning and backward error that underpins the rest of the thesis.

Chapters 2 and 3 look at new basis-specific algorithms for `GCD`, a fundamental polynomial operation that we attack in part via matrices. Changing basis from Bernstein or Lagrange is ill-conditioned, so we do not want to do that and instead we want to stay working in the given basis.

Chapter 4 expands the scope into matrix polynomials and looks at tools for a new linearization (that allows arbitrary bases).

Chapter 5 expands the scope again and investigates the case when matrices depend on a parameter and gives a new complexity result in the case all the variables appear linearly and that there are three or fewer parameters.

Chapter 6 is devoted to the concluding remarks and future work.

Bibliography

[1] R. M. Corless and N. Fillion. A Graduate Introduction to Numerical Methods: from the Viewpoint of Backward Error Analysis. In *Springer Publishing Company, Incorporated*, 2013.

[2] J. Von Zur Gathen, and J. Gerhard. Modern computer algebra. *Cambridge university press*, 2013.

[3] Carl B Boyer, and Uta C Merzbach. A history of mathematics. *John Wiley & Sons*, 2011.

[4] Dongming Wang and Li-Hong Zhi. Symbolic-numeric computation, *Springer Science & Business Media*, 2007.

Chapter 1

The Runge Example for Interpolation and Wilkinson's Examples for Rootfinding

The function $y = \frac{1}{1+25x^2}$ on $-1 \le x \le 1$, called the Runge example[1], is used in many numerical analysis textbooks to show why high-degree polynomial interpolation *using equally-spaced nodes* is bad: see for instance any of [16], [15], [4], [20], [1], [8], [11], [19], [12] and [17]. Some textbooks, even the iconic [14], do not emphasize the crucial qualification "using equally-spaced nodes"; or, perhaps, many readers skip over that[2] or don't retain it for other reasons, leaving only the false impression that high-degree interpolation is *always* bad.

Similarly, Wilkinson's first example polynomial [3]

$$p_{20}(x) = \prod_{k=1}^{20} (x - k) \tag{1.0.1}$$

is widely discussed as an example—maybe *the* canonical example—of polynomial perfidy, this time for rootfinding, not interpolation. As we will discuss below, both examples are better explained using the theory of *conditioning*, as developed for instance by Farouki and Rajan [10]. One considers a polynomial

$$p(x) = \sum_{k=0}^{n} c_k \phi_k(x) \tag{1.0.2}$$

expressed in some polynomial basis $\{\phi_k(x)\}_{k=0}^n$ for polynomials of degree at most n. The usual monomial basis $\phi_k(x) = x^k$ is the most common, but by no means best for all purposes. Farouki and Goodman [9] point out that the Bernstein basis $\phi_k(x) = \binom{n}{k} x^k (1-x)^{n-k}$ has the best conditioning in general, out of all polynomial bases satisfying

[1] Carl David Tolmé Runge (30 August 1856−3 January 1927) was a German mathematician, physicist, and spectroscopist.

[2] "tl;dr" as the kids say nowadays.

[3] James Hardy Wilkinson. Born: 27 September 1919 in Strood, Kent, England- Died: 5 October 1986 in Teddington, Middlesex, England. He worked with Turing in 1946. He won the Chauvenet Prize in 1970 for mathematical exposition for his paper "The Perfidious Polynomial" [22]. His book, "The Algebraic Eigenvalue Problem" was foundational for the field of numerical linear algebra.

$\phi_k(x) \geq 0$ on the interval $0 \leq x \leq 1$. Surprisingly, Corless and Watt [7] show Lagrange bases can be better; see also J. M. Carnicer and Y. Khiar [5].

We now discuss Farouki and Rajan's formulation. The idea is that one investigates the effects of small relative changes to the coefficients c_k, such as might arise from data error or perhaps approximation or computational error. The model is

$$p(x) + \Delta p(x) = \sum_{k=0}^{n} c_k(1+\delta_k)\phi_k(x) \tag{1.0.3}$$

where each $|\delta_k| \leq \varepsilon$, usually taken to be small. For instance, if $\varepsilon = 0.005$, each coefficient can be in error by no more than 0.5%. In particular, zero coefficients are not allowed to be disturbed at all.

Then,

$$|\Delta p(x)| = \left| \sum_{k=0}^{n} c_k(1+\delta_k)\phi_k(x) - \sum_{k=0}^{n} c_k\phi_k(x) \right| \tag{1.0.4}$$

$$= \left| \sum_{k=0}^{n} c_k\delta_k\phi_k(x) \right| . \tag{1.0.5}$$

By the triangle inequality, this is

$$\leq \sum_{k=0}^{n} |c_k\delta_k\phi_k(x)| \tag{1.0.6}$$

$$\leq \left(\sum_{k=0}^{n} |c_k||\phi_k(x)| \right) \max_{0\leq k\leq n} |\delta_k| \tag{1.0.7}$$

$$\leq B(x)\cdot\varepsilon \tag{1.0.8}$$

where

$$B(x) = \sum_{k=0}^{n} |c_k||\phi_k(x)| \tag{1.0.9}$$

serves, for each x, as a *condition number* for polynomial evaluation.
This is to be contrasted with the definition of "evaluation condition number" that arises when thinking of error Δx in the input: If x changes to $x+\Delta x$, then $y = f(x)$ changes to $y + \Delta y$ where calculus tells us that

$$\frac{\Delta y}{y} \doteq \frac{xf'(x)}{f(x)} \cdot \frac{\Delta x}{x} . \tag{1.0.10}$$

Here $C = xf'(x)/f(x)$ is the condition number, and instead of $B(x)$; but B and C measure the responses to different types of error (coefficients and input). We look at B here.

Remark 1.1. There are many theorems[4] in numerical analysis that say, effectively, that when evaluating equation (1.0.2) in IEEE floating-point arithmetic, the computed result is exactly of the form equation (1.0.3) for $|\delta_k| < Ku$ where K is a modest constant and

[4]See for instance those cited in Chapter 2 of [6], or equation (1.0.2) of [17] which gives a backward error result for an inner product; in that case the modest "constant" K is proportional to the dimension of the vector. See [18] for better, probabilistic error bounds.

u is the unit roundoff — in double precision, $2^{-53} \doteq 10^{-16}$. This is one motivation to study the effects of such perturbations, but there are others.

1.1 The Runge Example

If the equally-spaced nodes $x_k = -1 + 2k/n$, $k = 0, \ldots, n$, are used to interpolate a function with a single polynomial of degree at most n, and the basis functions

$$\ell_k(x) = \frac{\prod\limits_{\substack{j=0 \\ j\neq k}}^{n} (x - x_j)}{\prod\limits_{\substack{j=0 \\ j\neq k}}^{n} (x_k - x_j)} \tag{1.1.1}$$

which are the Lagrange interpolation basis functions, are used, then the coefficients are the values for the Runge function:

$$y_k = f(x_k) = \frac{1}{1 + 25x_k^2} \,. \tag{1.1.2}$$

Then the condition number of the interpolant is,

$$B(x) = \sum_{k=0}^{n} \frac{1}{1 + 25x_k^2} |\ell_k(x)| \,. \tag{1.1.3}$$

Choosing $n = 5, 8, 13, 21, 34, 55,$ and 89, we plot $B(x)$ on a logarithmic vertical scale for $-1 \leq x \leq 1$. The result is in Figure 1.1. The Maple code used to generate that figure is as follows (similar code using MATLAB can be provided, but the Maple code below avoids numerical issues in the construction of $B(x)$ by using exact rational arithmetic). We see that the maximum values for $B(x)$ occur near $x = \pm 1$, and that for any n there is an interval over which $B(x)$ is small.

```
Digits := 15:
Ns := [seq(combinat[fibonacci](k), k=5..11)]:
f := x -> 1/(1 + 25*x^2):

for N in Ns do
    tau := [seq(-1 + 2*k/N, k=0..N)]:
    rho := [seq(y[k], k=0..N)]:

    p := CurveFitting[PolynomialInterpolation](tau, rho, z,
   form=Lagrange):

    B := map(abs, p):
    BRunge := eval(B, [seq(y[k] = f(tau[k+1]), k=0..N)]):

    pl[N] := plots[logplot](BRunge, z=-1..1, color=black):

end do:

plots[display]([seq(pl[N], N in Ns)]);
```

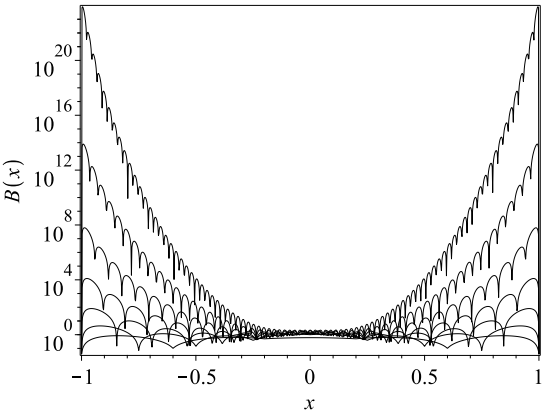

FIGURE 1.1: The condition number of the Runge example on equally-spaced nodes with degrees $n = 5, 8, 13, 21, 34, 55, 89$.

This experiment well illustrates that interpolation of the Runge example function on equally-spaced nodes is a bad idea. But, really, it is the nodes that are bad.

> "Generations of textbooks have warned readers that polynomial interpolation is dangerous. In fact, if the interpolation points are clustered and a stable algorithm is used, it is bulletproof."
>
> — L.N. Trefethen, [21]

For a full explanation of the Runge phenomenon, see Chapter 13 of [21].

1.1.1 The Runge Example with Chebyshev Nodes

If instead we use $x_k = \cos(\pi k/n)$, replacing the line

```
tau := [seq(-1 + 2*k/N, k=0..N)];
```

with

```
tau := [seq(evalf[2*Digits](cos(Pi*k/N)), k=0..N)];
```

then $B(x)$ climbs no higher than about 2. Indeed we can replace `plots[logplot]` by just `plot`. See Figure 1.2.

This is an improvement, for $n = 89$, of about a factor of 10^{22}. For a detailed exposition of why this works, and when, see [6] and the Chebfun project at www.chebfun.org.

1.1.2 Concluding remarks on the Runge example

An instructor of numerical analysis has to walk a tightrope: the students need to be taught caution (maybe bordering on paranoia) but they also need to learn when to trust their results. Learning to assess the sensitivity of their expression (as programmed) to realistic changes in data values is an important objective. The Runge example is a very clear case where these ideas can be usefully and thoroughly explored.

One can go further and replace $B(x)$ by its upper bound in terms of the Lebesgue function $B(x) \leq L(x)||c||_\infty$ where $L(x) = \sum_{k=0}^{n} |\phi_k(x)|^k$. This more general analysis is useful, as in [21], but loses in our opinion the chance to make a special retrospective diagnostic of the problem at hand. Moreover, there are cases where $B(x) \ll L(x)||c||_\infty$

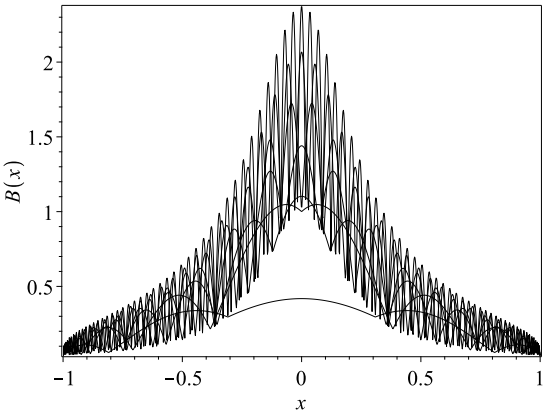

FIGURE 1.2: The Runge example with Chebyshev nodes with degrees $n = 5, 8, 13, 21, 34, 55, 89$.

and this overestimation could lead to the wrong conclusion.
The bad behavior of the Runge example shows up in other ways, notably in the ill-conditioning of the Vandermonde matrix on those nodes. But the Vandermonde matrix is ill-conditioned on Chebyshev nodes, too [3]; so that can't be the whole story. The explanation offered here seems more apt.

1.2 Wilkinson's First Polynomial

Let us now consider rootfinding. Suppose r is a simple zero of $p(x)$: That is, $p'(x) \neq 0$ and

$$0 = p(r) = \sum_{k=0}^{n} c_k \phi_k(r) \,. \tag{1.2.1}$$

Suppose $r + \Delta r$ is the corresponding zero of $p + \Delta p$. This really only makes sense if Δp is sufficiently small. Otherwise, the roots get mixed up. Then

$$0 = (p + \Delta p)(r + \Delta r) = p(r + \Delta r) + \Delta p(r + \Delta r) \tag{1.2.2}$$

$$\approx p(r) + p'(r)\Delta r + \Delta p(r) + \mathcal{O}(\Delta^2) \tag{1.2.3}$$

to first order; since $p(r) = 0$ also we have

$$p'(r)\Delta r \approx -\Delta p(r) \tag{1.2.4}$$

or

$$|\Delta r| \approx \left| \frac{-\Delta p(r)}{p'(r)} \right| \leq \frac{B(r) \cdot \varepsilon}{|p'(r)|} \tag{1.2.5}$$

where

$$B(r) = \sum_{k=0}^{n} |c_k||\phi_k(r)| \tag{1.2.6}$$

as before is the condition number. For nonzero roots, the number

$$A(r) = \left| \frac{rB(r)}{p'(r)} \right| \tag{1.2.7}$$

has $\left|\frac{\Delta r}{r}\right| \leq A(r)\varepsilon$ giving a kind of mixed relative/absolute conditioning. This analysis can be made more rigorous by using "pseudozeros" as follows. Define, for given $w_k \geq 0$ not all zero,

$$\Lambda_\varepsilon(p) := \left\{ z \,:\, \exists\, \Delta c_k \text{ with } |\Delta c_k| \leq w_k \varepsilon \text{ and } \sum_{k=0}^{n} (c_k + \Delta c_k)\phi_k(z) = 0 \right\} . \tag{1.2.8}$$

Normally, we take $w_k = |c_k|$ in which case we may write $\Delta c_k = c_k \delta_k$.
This is the set of all complex numbers that are zeros of "nearby" polynomials—nearby in the sense that we allow the coefficients to change. This definition is inconvenient to work with. Luckily, there is a useful theorem, which can be found, for instance, in [6, Theorem 5.3]; also see [13] and [2].

Theorem 1.2. *Given weights $w_k \geq 0$, not all zero, and a basis $\phi_k(z)$, define the weighted ε-pseudozero set of $p(z)$ as in equation (1.2.8). Suppose also that*

$$\delta p(z) = \sum_{k=0}^{n} \Delta c_k \phi_k(z).$$

Moreover, let

$$B(\lambda) = \sum_{k=0}^{n} w_k |\phi_k(\lambda)|.$$

Then the pseudozero set of $p(z)$ may be alternatively characterized as

$$\Lambda_\varepsilon(p) = \{ z \,:\, |p(z)| \leq B(z) \cdot \varepsilon \} = \left\{ z \,:\, \left|\frac{zp(z)}{p'(z)}\right| \leq \left|\frac{zB(z)}{p'(z)}\right| \varepsilon \right\} \tag{1.2.9}$$

This is again a condition number; the same one, as in equation (1.0.9) if $w_k = |c_k|$. Wilkinson's first polynomial is, with $N = 20$,

$$W_N(x) = \prod_{k=1}^{N} (x - k) \tag{1.2.10}$$

$$= (x-1)(x-2)(x-3)\cdots(x-N)\,. \tag{1.2.11}$$

In this form, it is "bulletproof". However, if we are so foolish as to expand it into its expression in the monomial basis, namely,

$$W_N(x) = x^N - \frac{1}{2}N(N+1)x^{N-1} + \cdots + (-1)^N \cdot N! \tag{1.2.12}$$

(for $N = 20$ this is $x^{20} - 210x^{19} + \cdots + (20!)$) and in this basis, $\phi_k = x^k$, the condition number for evaluation

$$B_N(x) = |x|^N + \frac{1}{2}N(N+1)|x|^{N-1} + \cdots + |N!|\,. \tag{1.2.13}$$

is very large. See Figure 1.3.

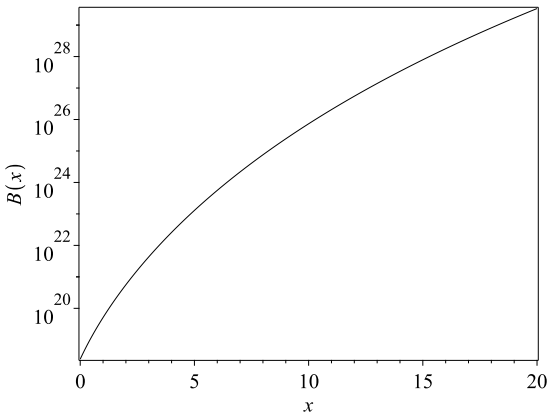

FIGURE 1.3: The condition number of Wilkinson's first polynomial ($N = 20$).

When we plot the condition number for root finding, $A(r) = \left|\frac{rB_N(r)}{W'_N(r)}\right|$, we find that for $N = 20$ (Wilkinson's original choice), the maximum value occurs at $r = 16$ and $rB_{20}(r)/|W'_{20}(r)| \approx 10^{16}$. See Figure 1.4.

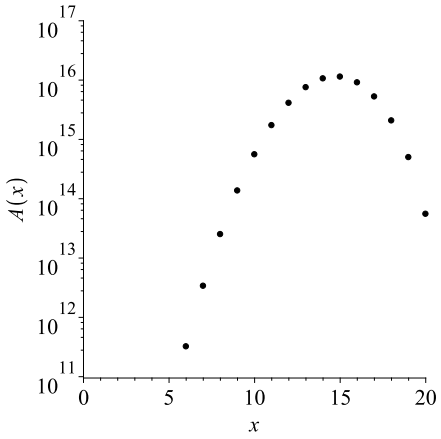

FIGURE 1.4: The condition number for rootfinding.

Working in single precision would give no figures of accuracy; double precision ($u \approx 10^{-16}$) also does not guarantee any accuracy. For $N = 30$ we find $\frac{rB_{30}(r)}{|W'_{30}(r)|} > 10^{21}$ sometimes; for $N = 40$ it's 10^{28}. Working with the monomial basis for this polynomial is surprisingly difficult. Wilkinson himself was surprised; the polynomial was intended to be a simple test problem for his program for the ACE computer. His investigations led to the modern theory of conditioning [22].

However, there's something a little unfair about the scaling: the interval $0 \le x \le 20$ when taken to the twentieth power covers quite a range of values. One wonders if matters can be improved by a simple change of variable.

1.2.1 The Scaled Wilkinson Polynomial

If we move the roots $1, 2, 3, \ldots, 20$ to the roots $-1 + 2k/21$, $k = 1 \ldots 30$, then they become symmetrically placed in $-1 < x < 1$, and this improves matters quite dramatically,

as we can see in Figure 1.5.

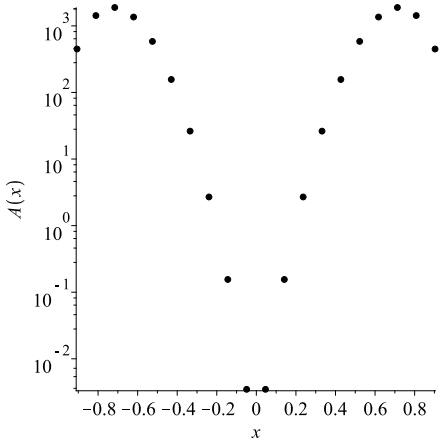

FIGURE 1.5: The condition number for the scaled Wilkinson polynomial, $A(r) = \left|\frac{rB_N(r)}{W_N'(r)}\right|$.

The condition number 10^{13} becomes just 10^3, and we have to go to $N = 60$ to get condition numbers as high as 10^{13}. The scaling seems to matter. However, nearly all of the improvement comes from the symmetry; W_N will be even if N is even, and odd if N is odd, and this means half the coefficients are zero and therefore not subject to (relative) perturbation.
If instead we scale to the interval $[0, 2]$ we have a different story: for roots $2 - 2k/21$ the condition number $B(x)$ reaches nearly the same heights as it did on $0 \le x \le 20$. See Figure 1.6. Similarly if we use $[0, 1]$. Thus we conclude that symmetry matters.

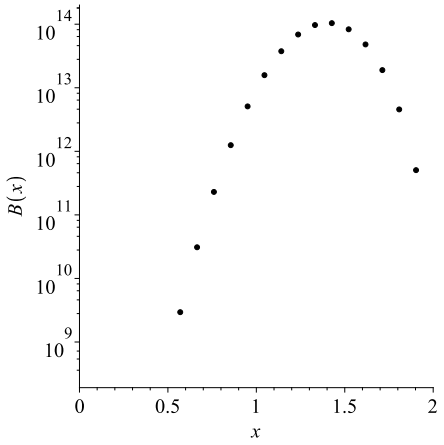

FIGURE 1.6: The condition number of scaled Wilkinson polynomial.

See Figure 1.7 for the pseudozeros of $W_N(x)$, where the contour levels are 10^{-14} and 10^{-18}. The roots are visibly changed by extremely tiny perturbations.

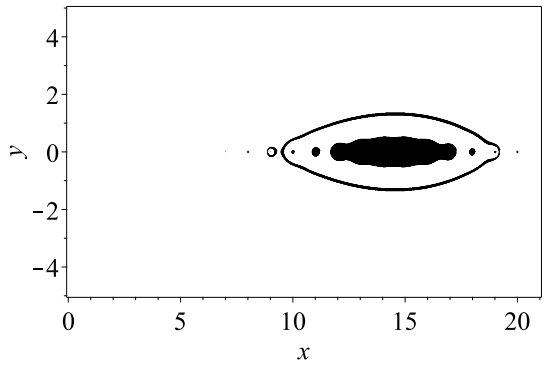

FIGURE 1.7: The pseudozeros of $W_N(x)$. The contour levels are 10^{-14} and 10^{-18}. The interior is blacked out because contours are difficult to draw at such sizes, in floating point arithmetic.

1.2.2 Wilkinson's Second Example Polynomial

The story of Wilkinson's second example is somehow more strange. The polynomial is

$$C_{20}(x) = \prod_{k=1}^{20} (x - 2^{-k}) \tag{1.2.14}$$

and the roots are $1/2, 1/4, 1/8, 1/16, \ldots, 1/2^{20}$. Wilkinson expected that the clustering of roots near zero would cause difficulty for his rootfinder, once the polynomial was expanded:

$$C_{20}(x) = x^{20} - \left(\sum_{k=1}^{20} \frac{1}{2^k} \right) x^{19} + \cdots + \prod_{k=1}^{20} 2^{-k} \,. \tag{1.2.15}$$

But his program had no difficulty at all! This is because the monomial basis is, in fact, quite well-conditioned near $x = 0$, and the condition number for this polynomial can be seen in Figure 1.8 on $0 \le x \le 1$.

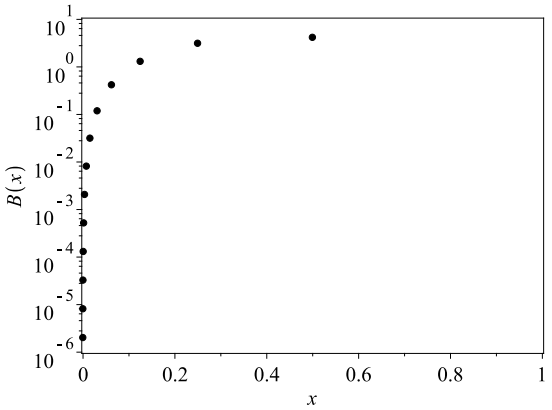

FIGURE 1.8: The condition number for Wilkinson's second example polynomial (C_{20}). In contrast to his first test problem, it is well-conditioned.

In contrast, the condition number for evaluation using the Lagrange basis on equally-spaced nodes in $[0,1]$, plus either $x_0 = 0$ or $x_0 = 1$, is horrible: for $N = 20$ it is already 10^{48}, see Figure 1.9.

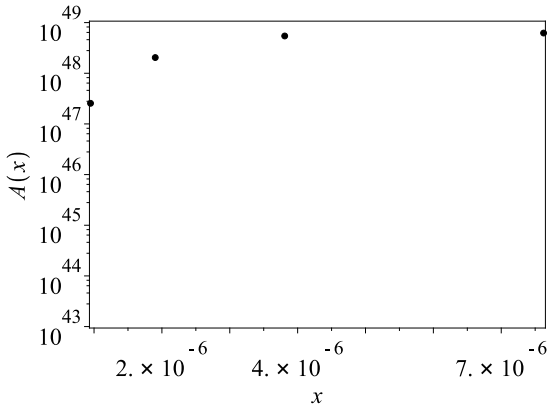

FIGURE 1.9: A portion of the condition number of C_{20} in the Lagrange basis on the nodes $k/20$, $0 \le k \le 20$.

This computation conforms to Wilkinson's intuition that things can go wrong if roots are clustered. Also we can see the pseudozeros of C_{20} in Figure 1.10. The required perturbations needed to make visible changes are quite large: these roots are not very sensitive to changes in the monomial basis coefficients.

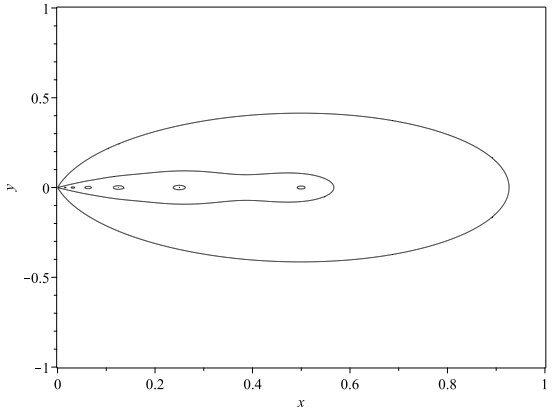

FIGURE 1.10: The pseudozeros of C_{20}. The contour levels are 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-6} and 10^{-8}.

Another way to see this is to look at a problem where the roots are clustered at 1, not at 0:

$$S_{20} = \prod_{k=1}^{N} \left(x - (1 - 2^{-k}) \right) = \sum_{k=0}^{N} s_k x^k \tag{1.2.16}$$

In this case the condition number is presented in Figure 1.11, and is huge. This polynomial is very sensitive to changes in the monomial basis coefficients.

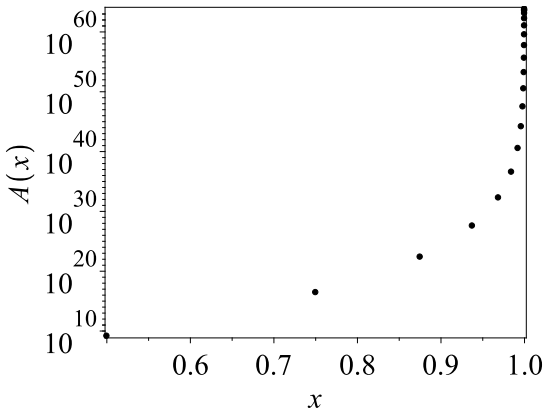

FIGURE 1.11: The condition number of S_{20}.

The condition number for evaluation using a Lagrange basis for S_{20} is shown in Figure 1.12 (zoomed in for emphasis). Here the Lagrange basis is also very sensitive.

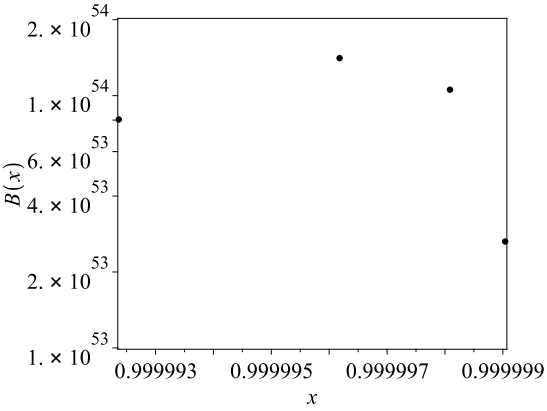

FIGURE 1.12: The condition number of S_{20} in a Lagrange basis.

Consider also the pseudozeros of S_{20} in Figure 1.13. The contour levels in Figure 1.13 are (from the outside in) 10^{-4}, 10^{-6}, 10^{-8}, 10^{-10} and 10^{-15}. In order to see a better view of the pseudozeros, let's consider the first contour, 10^{-4}, which is the biggest curve in Figure 1.13. We know that

$$S_{20} + \Delta S = \sum_{k=0}^{20} s_k(1 + \delta_k)x^k \tag{1.2.17}$$

where $\Delta S = s_0\delta_0 + s_1\delta_1 x + s_2\delta_2 x^2 + \cdots + s_{20}\delta_{20}x^{20}$. Now if we choose a point between contour levels 10^{-4} and 10^{-6}, for example $p = 3 - 1.5i$, we can see that p is a zero of some $S_{20} + \Delta S(x)$ with all coefficients of ΔS that have $|\delta_k| < 10^{-4}$. These are all small relative perturbations, that means everything inside the contour level 10^{-4} is a zero of a polynomial that is reasonably close to S_{20}. This is somehow backward error. So everything inside the contour level 10^{-4} is a zero of a polynomial closer to S_{20} (in this sense) than 10^{-4}. Everything inside the contour level 10^{-6} is a zero of a polynomial closer that 10^{-6} to S_{20}, and so on.

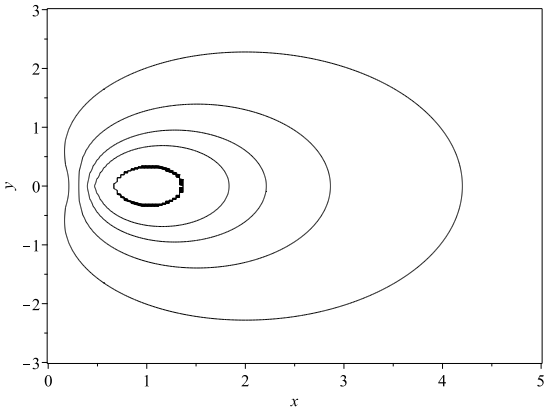

FIGURE 1.13: The pseudozeros of S_{20}. The contour levels are 10^{-4}, 10^{-6}, 10^{-8}, 10^{-10} and 10^{-15}.

Notice that the innermost contour, corresponding to 10^{-15}, is visible to the eye. This means that trivial (unit roundoff level in double precision) changes in the coefficients make visible changes in the root.

1.2.3 Concluding remarks on the Wilkinson rootfinding examples

The first example polynomial, $\prod_{k=1}^{20}(x-k)$, is nearly universally known as a surprising example. Yet there are very few places where one sees an elementary exposition of Wilkinson's theory of conditioning using this example, which is itself surprising because the theory was essentially born from this example. We have here illustrated Wilkinson's theory, as refined by Farouki and Rajan, for the students.

> "For accidental historical reasons therefore backward error analysis is always introduced in connexion with matrix problems. In my opinion the ideas involved are much more readily absorbed if they are presented in connexion with polynomial equations. Perhaps the fairest comment would be that polynomial equations narrowly missed serving once again in their historical didactic role and rounding error analysis would have developed in a more satisfactory way if they had not."
>
> — James H. Wilkinson, [22]

1.3 A final word for the instructor

Backward error analysis is difficult at first for some kinds of students. The conceptual problem is that people are trained to think of mathematical problems as being exact; some indeed are, but many come from physical situations and are only models, with uncertain data. The success of BEA for floating point is to put rounding errors on the same footing as data or modeling errors, which have to be studied anyway. This is true even if the equations are solved exactly, by using computer algebra! The conditioning theory for polynomials discussed here allow this to be done quite flexibly, and are useful part of the analyst's repertoire. Students need to know this.

Bibliography

[1] F. Acton, *Numerical Methods that Work*, MAA spectrum, Mathematical Association of America, 1990, https://books.google.ca/books?id=cGnSMGSE5Y4C.

[2] A. Amiraslani, *New Algorithms for Matrices, Polynomials and Matrix Polynomials*, PhD thesis, Western University, 2006.

[3] B. Beckermann, *The condition number of real vandermonde, krylov and positive definite hankel matrices*, Numerische Mathematik, 85 (2000), pp. 553–577.

[4] R. Burden, D. Faires, and A. Burden, *Numerical analysis*, Cengage Learning, Boston, MA, tenth edition. ed., 2016.

[5] J. Carnicer, Y. Khiar, and J. Peña, *Optimal stability of the Lagrange formula and conditioning of the Newton formula*, Journal of Approximation Theory, (2017).

[6] R. M. Corless and N. Fillion, *A Graduate Introduction to Numerical Methods: from the Viewpoint of Backward Error Analysis*, Springer Publishing Company, Incorporated, 2013, https://doi.org/10.1007/978-1-4614-8453-0.

[7] R. M. Corless and S. M. Watt, *Bernstein bases are optimal, but, sometimes, lagrange bases are better*, in In Proceedings of SYNASC, Timisoara, MIRTON Press, 2004, pp. 141–153.

[8] T. Driscoll and R. Braun, *Fundamentals of Numerical Computation*, Other Titles in Applied Mathematics, Society for Industrial and Applied Mathematics, 2017, https://books.google.ca/books?id=WdJEDwAAQBAJ.

[9] R. Farouki and T. Goodman, *On the optimal stability of the Bernstein basis*, Mathematics of Computation of the American Mathematical Society, 65 (1996), pp. 1553–1566.

[10] R. Farouki and V. Rajan, *On the numerical condition of polynomials in Bernstein form*, Computer Aided Geometric Design, 4 (1987), pp. 191–216.

[11] W. Gautschi, *Numerical Analysis*, SpringerLink : Bücher, Birkhäuser Boston, 2011, https://books.google.ca/books?id=-fgjJF9yAIwC.

[12] T. Gowers, J. Barrow-Green, and I. Leader, *The Princeton Companion to Mathematics*, Princeton University Press, 2010, https://books.google.ca/books?id=ZOfUsvemJDMC.

[13] K. Green and T. Wagenknecht, *Pseudospectra and delay differential equations*, Journal of Computational and Applied Mathematics, 196 (2006), pp. 567 – 578.

[14] R. Hamming, *Numerical Methods for Scientists and Engineers*, Dover Books on Mathematics, Dover Publications, 2012, https://books.google.ca/books?id=Z2owE_0LQukC.

[15] P. Henrici, *Elements of numerical analysis*, tech. report, 1964.

[16] P. Henrici, *Essentials of numerical analysis with pocket calculator demonstrations*, John Wiley & Sons, Inc., 1982.

[17] N. Higham, *Accuracy and Stability of Numerical Algorithms: Second Edition*, Other Titles in Applied Mathematics, Society for Industrial and Applied Mathematics, 2002, https://books.google.ca/books?id=5tv3HdF-0N8C.

[18] N. J. Higham and T. Mary, *A new approach to probabilistic rounding error analysis*, (2018).

[19] D. Kahaner, C. Moler, and S. Nash, *Numerical methods and software*, Prentice-Hall series in computational mathematics, Prentice Hall, 1989, https://books.google.ca/books?id=jipEAQAAIAAJ.

[20] T. Sauer, *Numerical Analysis*, Always learning, Pearson, 2011, https://books.google.ca/books?id=81kEYAAACAAJ.

[21] L. N. Trefethen, *Approximation theory and approximation practice*, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2013.

[22] J. H. Wilkinson, *The perfidious polynomial*, Studies in Numerical Analysis, 24 (1984), pp. 1–28.

Chapter 2

Approximate GCD in a Bernstein basis

2.1 Introduction

In general, finding the Greatest Common Divisor (GCD) of two exactly-known univariate polynomials is a well understood problem. However, it is also known that the GCD problem for *noisy* polynomials (polynomials with errors in their coefficients) is an ill-posed problem. More precisely, a small error in coefficients of polynomials P and Q with a non-trivial GCD generically leads to a trivial GCD. As an example of such situation, suppose P and Q are non constant polynomials such that $P|Q$ (P divides Q), then $\gcd(P, Q) = P$. Now for any $\epsilon > 0$, $\gcd(P, Q+\epsilon)$ is a constant, since if $\gcd(P, Q+\epsilon) = g$, then $g|Q + \epsilon - Q = \epsilon$. This clearly shows that the GCD problem is an ill-posed one. We note that the choice of basis makes no difference to this difficulty.

At this point we have a good motivation to define something which can play a similar role to the GCD of two given polynomials which is instead well-conditioned. The idea is to define an *approximate GCD* [5]. There are various definitions for approximate GCD which are used by different authors. All these definitions respect "closeness" and "divisibility" in some sense.

In this paper an approximate GCD of a pair of polynomials P and Q is the exact GCD of a corresponding pair of polynomials $\tilde{P}$ and $\tilde{Q}$ where P and $\tilde{P}$ are "close" with respect to a specific metric, and similarly for Q and $\tilde{Q}$ (see Definition 2.1).

Finding the GCD of two given polynomials is an elementary operation needed for many algebraic computations. Although in most applications the polynomials are given in the power basis, there are cases where the input is given in other bases such as the Bernstein basis. One important example of such a problem is finding intersection points of Bézier curves, which is usually presented in a Bernstein basis. For computing the intersections of Bézier curves and surfaces the Bernstein resultant and GCD in the Bernstein basis comes in handy (see [3]).

One way to deal with polynomials in Bernstein bases is to convert them into the power basis. In practice poor stability of conversion from one basis to another and poor conditioning of the power basis essentially cancel the benefit one might get by using conversion to the simpler basis (see [10]).

The Bernstein basis is an interesting one for various algebraic computations, for instance, see [19], [21]. There are many interesting results in approximate GCD including but not limited to [1], [5], [2], [20], [26], [8], [17], [18] and [16]. In [27], the author has introduced a modification of the algorithm given by Corless, Gianni, Trager and Watt in [7], to compute the approximate GCD in the power basis.

Winkler and Yang in [25] give an estimate of the degree of an approximate GCD of two polynomials in a Bernstein basis. Their approach is based on computations using resultant matrices. More precisely, they use the singular value decomposition of Sylvester and Bézout resultant matrices. We do not follow the approach of Winkler and Yang here, because they essentially convert to a power basis. Owing to the difference of results we do not give a comparison of our algorithm with the results of [25].

Our approach is mainly to follow the ideas introduced by Victor Y. Pan in [21], working in the power basis. In distinction to the other known algorithms for approximate GCD, Pan's method does not algebraically compute a degree for an approximate GCD first. Instead it works in a reverse way. In [21] the author assumes the roots of polynomials P and Q are given as inputs. Having the roots in hand the algorithm generates a bipartite graph where one set of nodes contains the roots of P and the other contains the roots of Q. The criterion for defining the set of edges is based on Euclidean distances of roots. When the graph is defined completely, a matching algorithm will be applied. Using the obtained matching, a polynomial D with roots as averages of paired close roots will be produced which is considered to be an approximate GCD. The last step is to use the roots of D to replace the corresponding roots in P and Q to get $\tilde{P}$ and $\tilde{Q}$ as close polynomials.

In this paper we introduce an algorithm for computing approximate GCD in the Bernstein basis which relies on the above idea. For us the inputs are the coefficient vectors of P and Q. We use the correspondence between the roots of a polynomial f in a Bernstein basis and generalized eigenvalues of a corresponding matrix pencil (A_f, B_f). This idea for finding the roots of f was first used in [14]. Then by finding the generalized eigenvalues we get the roots of P and Q (see [14, Section 2.3]). Using the roots and similar methods to [21], we form a bipartite graph and then we apply the maximum matching algorithm by Hopcroft and Karp [13] to get a maximum matching. Having the matching, the algorithm forms a polynomial which is considered as an approximate GCD of P and Q. The last step is to construct $\tilde{P}$ and $\tilde{Q}$ for which we apply a generalization of the method used in [6, Example 6.10] (see Section 3.6).

Note that our algorithm, like that of Victor Y. Pan, does almost the reverse of the well-known algorithms for approximate GCD. Usually the algebraic methods do not try to find the roots. In [21] Pan assumes the polynomials are represented by their roots. In our case we do not start with this assumption. Instead, by computing the roots we can then apply Pan's method.

The second section of this paper is provided some background for concrete computations with polynomials in Bernstein bases which is needed for our purposes. The third section present a method to construct a corresponding pair of polynomials to a given pair (P, Q). More precisely, this section generalizes the method mentioned in [6, Example 6.10] (which is introduced for power basis) in the Bernstein basis. The fourth section introduces a new algorithm for finding an approximate GCD. In the final section we present numerical results based on our method.

2.2 Preliminaries

The Bernstein polynomials on the interval $0 \leq x \leq 1$ are defined as

$$B_k^n(x) = \binom{n}{k} x^k (1-x)^{n-k} \tag{2.2.1}$$

for $k = 0, \ldots, n$, where the binomial coefficient is as usual

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} . \tag{2.2.2}$$

More generally, in the interval $a \leq x \leq b$ (where $a < b$) we define

$$B^n_{a,b,k}(x) := \binom{n}{k} \frac{(x-a)^k (b-x)^{n-k}}{(b-a)^n} . \tag{2.2.3}$$

When there is no risk of confusion we may simply write B^n_k's for the $0 \leq x \leq 1$ case. We suppose henceforth that $P(x)$ and $Q(x)$ are given in a Bernstein basis.

There are various definitions for approximate GCD. The main idea behind all of them is to find "interesting" polynomials $\tilde{P}$ and $\tilde{Q}$ close to P and Q and use $\gcd(\tilde{P}, \tilde{Q})$ as the approximate GCD of P and Q. However, there are multiple ways of defining both "interest" and "closeness". To be more formal, consider the following weighted norm, for a vector v

$$\|v\|_{\alpha,r} = \left(\sum_{k=1}^{n} |\alpha_k v_k|^r \right)^{1/r} \tag{2.2.4}$$

for a given weight vector $\alpha \neq 0$ and a positive integer r or ∞. The map $\rho(u, v) = \|u - v\|_{\alpha,r}$ is a metric and we use this metric to compare the coefficient vectors of P and Q.

In this paper we define an approximate GCD using the above metric or indeed any fixed semimetric. More precisely, we define the *pseudogcd* set for the pair P and Q as

$$A_\rho = \{ g(x) \mid \exists \tilde{P}, \tilde{Q} \text{ with } \rho(P, \tilde{P}) \leq \sigma, \rho(Q, \tilde{Q}) \leq \sigma \text{ and } g(x) = \gcd(\tilde{P}, \tilde{Q}) \} .$$

Let

$$d = \max_{g \in A_\rho} \deg(g(x)) . \tag{2.2.5}$$

Definition 2.1. An approximate GCD for P, Q, which is denoted by $\operatorname{agcd}^\sigma_\rho(P, Q)$, is $G(x) \in A_\rho$ where $\deg(G) = d$ and $\rho(P, \tilde{P})$ and $\rho(Q, \tilde{Q})$ are simultaneously minimal in some sense. For definiteness, we suppose that the maximum of these two quantities is minimized.

In Section 2.2.3 we will define another (semi) metric, that uses roots. In Section 2.4 we will see that the parameter σ helps us to find approximate polynomials such that the common roots of $\tilde{P}$ and $\tilde{Q}$ have at most distance (for a specific metric) σ to the associated roots of P and Q (see Section 2.4 for more details).

2.2.1 Finding roots of a polynomial in a Bernstein basis

In this section we recount the numerically stable method introduced by Guðbjörn Jónsson for finding roots of a given polynomial in a Bernstein basis. We only state the method without discussing in detail its stability and we refer the reader to [14] and [15] for more details.

Consider a polynomial

$$P(x) = \sum_{i=0}^{n} a_i B^n_i(x) \tag{2.2.6}$$

in a Bernstein basis where the a_i's are real scalars. We want to find the roots of $P(x)$ by constructing its companion pencil. In [14] Jónsson showed that this problem is equivalent

to solving the following generalized eigenvalue problem. That is, the roots of $P(x)$ are the generalized eigenvalues of the corresponding companion pencil to the pair

$$\mathbf{A}_P = \begin{bmatrix} -a_{n-1} & -a_{n-2} & \cdots & -a_1 & -a_0 \\ 1 & 0 & & & \\ & 1 & 0 & & \\ & & \ddots & \ddots & \\ & & & 1 & 0 \end{bmatrix}, \quad \mathbf{B}_P = \begin{bmatrix} -a_{n-1} + \frac{a_n}{n} & -a_{n-2} & \cdots & -a_1 & -a_0 \\ 1 & \frac{2}{n-1} & & & \\ & 1 & \frac{3}{n-2} & & \\ & & \ddots & \ddots & \\ & & & 1 & \frac{n}{1} \end{bmatrix}.$$

That is, $P(x) = \det(x\mathbf{B}_P - \mathbf{A}_P)$. In [14], the author showed that the above method is numerically stable.

Theorem 2.2. *[14, Section 2.3] Assume $P(x)$, $\boldsymbol{A}_P$ and $\boldsymbol{B}_P$ are defined as above. z is a root of $P(x)$ if and only if it is a generalized eigenvalue for the pair $(\boldsymbol{A}_P, \boldsymbol{B}_P)$.*

Proof. We show

$$P(z) = 0 \quad \Leftrightarrow \quad (z\mathbf{B}_P - \mathbf{A}_P) \begin{bmatrix} B_{n-1}^n(z)(\frac{1}{1-z}) \\ \vdots \\ B_1^n(z)(\frac{1}{1-z}) \\ B_0^n(z)(\frac{1}{1-z}) \end{bmatrix} = 0\,. \tag{2.2.7}$$

We will show that all the entries of

$$(z\mathbf{B}_P - \mathbf{A}_P) \begin{bmatrix} B_{n-1}^n(z)(\frac{1}{1-z}) \\ \vdots \\ B_1^n(z)(\frac{1}{1-z}) \\ B_0^n(z)(\frac{1}{1-z}) \end{bmatrix} \tag{2.2.8}$$

are zero except for possibly the first entry:

$$(z-1)B_1^n(z)(\frac{1}{z-1}) + nzB_0^n(z)(\frac{1}{z-1}) \tag{2.2.9}$$

since $B_1^n(z) = nz(1-z)^{n-1}$ and $B_0^n(z) = z^0(1-z)^n$ if $n \geq 1$, so equation (2.2.9) can be written as

$$-nz(1-z)^{n-1} + nz\frac{(1-z)^n}{(1-z)} = -nz(1-z)^{n-1} + nz(1-z)^{n-1} = 0\,. \tag{2.2.10}$$

Now for k-th entry:

$$\frac{(z-1)}{(1-z)}B_{n-k}^n(z) + \frac{k+1}{n-k}\frac{z}{(1-z)}B_{n-k-1}^n(z) \tag{2.2.11}$$

Again we can replace $B^n_{n-k}(z)$ and $B^n_{n-k-1}(z)$ by their definitions. We find that equation (2.2.11) can be written as

$$\begin{aligned}\frac{(z-1)}{(1-z)}\binom{n}{n-k}z^{n-k}(1-z)^{n-n+k}+\frac{k+1}{n-k}\frac{z}{(1-z)}\binom{n}{n-(k+1)}z^{n-k-1}(1-z)^{n-(n-k-1)}\\ =-\binom{n}{n-k}z^{n-k}(1-z)^k+\frac{k+1}{n-k}\frac{n!}{(n-(k+1))!(k+1)!}z^{n-k}(1-z)^k\\ =\frac{n!}{(n-k)!k!}z^{n-k}(1-z)^k+\frac{n!}{(n-k)(n-(k+1))!k!}z^{n-k}(1-z)^k=0\,. \quad (2.2.12)\end{aligned}$$

Finally, the first entry of equation (2.2.8) is

$$\frac{za_n}{n(1-z)}B^n_{n-1}(z)+\frac{a_{n-1}(z-1)}{(1-z)}B^n_{n-1}(z)+\sum_{i=0}^{n-2}a_iB^n_{n-1}(z) \quad (2.2.13)$$

In order to simplify the equation (2.2.13), we use the definition of $B^n_{n-1}(z)$ as follows:

$$\frac{za_n}{n(1-z)}B^n_{n-1}(z)=\frac{z}{n}\binom{n}{n-1}z^{n-1}\frac{1-z}{1-z}=\frac{z^na_n}{n}\binom{n}{n-1}=a_nB^n_n(z) \quad (2.2.14)$$

So the equation (2.2.13) can be written as:

$$a_nB^n_n(z)+(a_{n-1})B^n_{n-1}(z)+\sum_{i=0}^{n-2}a_iB^n_{n-1}(z) \quad (2.2.15)$$

This is just $P(z)$ and so

$$(z\mathbf{B}_P-\mathbf{A}_P)\begin{bmatrix}B^n_{n-1}(z)(\frac{1}{1-z})\\ \vdots\\ B^n_1(z)(\frac{1}{1-z})\\ B^n_0(z)(\frac{1}{1-z})\end{bmatrix}=\begin{bmatrix}0\\ \vdots\\ 0\\ 0\end{bmatrix} \quad (2.2.16)$$

if and only if $P(z)=0$. □

This pencil (or rather its transpose) has been implemented in Maple since 2004.

Example 2.1. *Suppose $P(x)$ is given by its list of coefficients*

$$[42.336, 23.058, 11.730, 5.377, 2.024] \quad (2.2.17)$$

Then by using Theorem 2.2, we can find the roots of $P(x)$ by finding the eigenvalues of its corresponding companion pencil namely:

$$\boldsymbol{A}_P:=\begin{bmatrix}-5.377 & -11.73 & -23.058 & -42.336\\ 1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & 0\end{bmatrix} \quad (2.2.18)$$

and

$$\boldsymbol{B}_P := \begin{bmatrix} -4.871 & -11.73 & -23.058 & -42.336 \\ 1 & .6666666666 & 0 & 0 \\ 0 & 1 & 1.5 & 0 \\ 0 & 0 & 1 & 4 \end{bmatrix} \tag{2.2.19}$$

Now if we solve the generalized eigenvalue problem using Maple for pair of $(\boldsymbol{A}_P, \boldsymbol{B}_P)$ *we get:*

$$[\ 5.59999999999989, 3.00000000000002, 2.1, 1.2\ ] \tag{2.2.20}$$

Computing residuals, we have exactly[1] $P(1.2) = 0$, $P(2.1) = 0$, $P(3) = 0$, *and* $P(5.6) = 0$ *using de Casteljau's algorithm (see Section 2.2.4).*

2.2.2 Clustering the roots

In this brief section we discuss the problem of having multiplicities greater than 1 for roots of our polynomials. Since we are dealing with approximate roots, for an specific root r of multiplicity m, we get $r_1, \ldots, r_m$ where $|r - r_i| \leq \sigma$ for $\sigma \geq 0$. Our goal in this section is to recognize the *cluster*, $\{r_1, \ldots, r_m\}$, for a root r as $\tilde{r}^m$ where $|\tilde{r} - r| \leq \sigma$ in a constructive way.

Assume a polynomial f is given by its roots as $f(x) = \prod_{i=1}^{n}(x - r_i)$. Our goal is to write $f(x) = \prod_{i=1}^{s}(x - t_i)^{d_i}$ such that $(x - t_i) \nmid f(x)/(x - t_i)^{d_i}$. In other words, d_i's are multiplicities of t_i's. In order to do so we need a parameter σ to compare the roots. If $|r_i - r_j| \leq \sigma$ then we replace both r_i and r_j with their average.

For our purposes, even the naive method, *i.e.* computing distances of all roots, works. This idea is presented as Algorithm 1. It is worth mentioning that for practical purposes a slightly better way might be a modification of the well known divide and conquer algorithm for solving the closest pair problem in plane [28, Section 33.4].

2.2.3 The root marriage problem

The goal of this section is to provide an algorithmic solution for solving the following problem:

The Root Marriage Problem (RMP): Assume P and Q are polynomials given by their roots. For a given $\sigma > 0$, for each root r of P, (if it is possible) find a unique root of Q, say s, such that $|r - s| \leq \sigma$.

A solution to the RMP can be achieved by means of graph theory algorithms. We recall that a maximum matching for a bipartite graph (V, E), is $M \subseteq E$ with two properties:

- every node $v \in V$ appears as an end point of an edge in M at most once.
- M has the maximum size among the subsets of E satisfying the previous condition.

We invite the reader to consult [4] and [24] for more details on maximum matching.

There are various algorithms for solving the maximum matching problem in a graph. Micali and Vazirani's matching algorithm is probably the most well-known. However there are more specific algorithms for different classes of graphs. In this paper, as in [21], we use the Hopcroft-Karp algorithm for solving the maximum matching problem in a bipartite graph which has a complexity of $O((m + n)\sqrt{n})$ operations.

[1] In some sense the exactness is accidental; the computed residual is itself subject to rounding errors. See [6] for a backward error explanation of how this can happen.

Algorithm 1 `ClusterRoots`(P, σ)

Input: P is a list of roots
Output: $[(\alpha_1, d_1), \ldots, (\alpha_m, d_m)]$ where α_i is considered as a root with multiplicity d_i

```
temp ← EmptyList
C ← EmptyList
p ← size(P)
i ← 1
while i ≤ p do
    append(temp, P[i])
    j ← i + 1
    while j ≤ p do
        if |P[i] − P[j]| ≤ s then
            append(temp, P[j])
            remove(P, j)
            p ← p − 1
        else
            j ← j + 1
    i ← i + 1
    append(C, [Mean(temp), size(temp)])
return C
```

Now we have enough tools for solving the RMP. The idea is to reduce the RMP to a maximum matching problem. In order to do so we have to associate a bipartite graph to a pair of polynomials P and Q. For a positive real number σ, let $G^{\sigma}_{P,Q} = (G^{\sigma}_P \cup G^{\sigma}_Q, E^{\sigma}_{P,Q})$ where

- $G^{\sigma}_P =$ `ClusterRoots`(the set of roots of P, σ),
- $G^{\sigma}_Q =$ `ClusterRoots`(the set of roots of Q, σ),
- $E^{\sigma}_{P,Q} = \Big\{ (\{r, s\},\ \min(d_t, d_s)) :\ r \in G^{\sigma}_P,\ \text{with multiplicity}\ \ d_r,\ s \in G^{\sigma}_Q$ with multiplicity $d_s,\ \Big|r[1] - s[1]\Big| \leq \sigma \Big\}$

Assuming we have access to the roots of polynomials, it is not hard to see that there is a naive algorithm to construct $G^{\sigma}_{P,Q}$ for a given $\sigma > 0$. Indeed it can be done by performing $O(n^2)$ operations to check the distances of roots where n is the larger degree of the given pair of polynomials.

The last step to solve the RMP is to apply the Hopcroft-Karp algorithm on $G^{\sigma}_{P,Q}$ to get a maximum matching. The complexity of this algorithm is $O(n^{\frac{5}{2}})$ which is the dominant term in the total cost. Hence we can solve RMP in time $O(n^{\frac{5}{2}})$.

As was stated in Section 2.2, we present a semi-metric which works with polynomial roots in this section. For two polynomials R and T, assume $m \leq n$ and $\{r_1, \ldots, r_m\}$ and $\{t_1, \ldots, t_n\}$ are respectively the sets of roots of R and T. Moreover assume S_n is the set of all permutations of $\{1, \ldots, n\}$. We define

$$\rho(R, T) = \min_{\tau \in S_n} \| [r_1 - t_{\tau(1)}, \ldots, r_m - t_{\tau(m)}] \|_{\alpha, r},$$

where α and r are as before.

Remark 2.3. The cost of computing this semi-metric by this definition is $O(n!)$, and therefore prohibitive. However, once a matching has been found then

$$\rho(R,T) = \| [r_1 - s_{\text{match}(1)}, r_2 - s_{\text{match}(2)}, \ldots, r_m - s_{\text{match}(m)}] \|_{\alpha,r}$$

where the notation $s_{\text{match}(k)}$ indicates the root found by the matching algorithm that matches r_k.

2.2.4 de Casteljau's Algorithm

Another component of our algorithm is a method which enables us to evaluate a given polynomial in a Bernstein basis at a given point. There are various methods for doing that. One of the most popular algorithms, for its convenience in Computer Aided Geometric Design (CAGD) applications and its numerical stability [9], is de Casteljau's algorithm which for convenience here is presented as Algorithm 2. We note that

Algorithm 2 `de Casteljau's Algorithm`

Input: C: a list of coefficients of a polynomial $P(x)$ of degree n in a Bernstein basis of size $n+1$
α: a point
Output: $P(\alpha)$
1: $c_{0,j} \leftarrow C_j$ for $j = 0 \ldots n$.
2: recursively define
$c_{i,j} \leftarrow (1-\alpha) \cdot c_{i-1,j} + \alpha \cdot c_{i-1,j+1}$.
for $i = 1 \ldots n$ and $j = 1 \ldots n-i$.
3: return $c_{n,0}$.

the above algorithm uses $O(n^2)$ operations for computing $P(\alpha)$. In contrast, Horner's algorithm for the power basis, Taylor polynomials, or the Newton basis, and the Clenshaw algorithm for orthogonal polynomials, and the barycentric forms[2] for Lagrange and Hermite interpolational basis cost $O(n)$ operations.

2.3 Computing Approximate Polynomials

This section is a generalization of [6, Example 6.10] in Bernstein bases. The idea behind the algorithm is to create a linear system from coefficients of a given polynomial and the values of the polynomial at the approximate roots.

Now assume

$$P(x) = \sum_{i=0}^{n} p_i B_i^n(x) \tag{2.3.1}$$

is given with $\alpha_1, \ldots, \alpha_t$ as its approximate roots with multiplicities d_i. Our aim is to find

$$\tilde{P}(x) = (P + \Delta P)(x) \tag{2.3.2}$$

where

$$\Delta P(x) = \sum_{i=0}^{n} (\Delta p_i) B_i^n(x) \tag{2.3.3}$$

[2] Assuming that the barycentric weights are precomputed.

so that the set $\{\alpha_1, \ldots, \alpha_t\}$ appears as exact roots of $\tilde{P}$ with multiplicities d_i respectively. On the other hand, we do want to have some control on the coefficients in the sense that the new coefficients are related to the previous ones. Defining $\Delta p_i = p_i \delta p_i$ (which assumes p_i's are non-zero) yields

$$\tilde{P}(x) = \sum_{i=0}^{n} (p_i + p_i \delta p_i) B_i^n(x) \tag{2.3.4}$$

Representing P as above, we want to find $\{\delta p_i\}_{i=0}^n$. It is worth mentioning that with our assumptions, since perturbations of each coefficient, p_i of P are proportional to itself, if $p_i = 0$ then $\Delta p_i = 0$. In other words we have assumed zero coefficients in P will not be perturbed.

In order to satisfy the conditions of our problem we have

$$\tilde{P}(\alpha_j) = \sum_{i=0}^{n} (p_i + p_i \delta p_i) B_i^n(\alpha_j) = 0\,, \tag{2.3.5}$$

for $j = 1, \ldots, t$. Hence

$$\tilde{P}(\alpha_j) = \sum_{i=0}^{n} p_i B_i^n(\alpha_j) + \sum_{i=0}^{n} p_i \delta p_i B_i^n(\alpha_j) = 0\,, \tag{2.3.6}$$

or equivalently

$$\sum_{i=0}^{n} p_i \delta p_i B_i^n(\alpha_j) = -P(\alpha_j)\,, \tag{2.3.7}$$

Having the multiplicities, we also want the approximate polynomial $\tilde{P}$ to respect multiplicities. More precisely, for α_j, a root of P of multiplicity d_j, we expect that α_j has multiplicity d_j as a root of $\tilde{P}$. As usual we can state this fact by means of derivatives of $\tilde{P}$. We want

$$\tilde{P}^{(k)}(\alpha_j) = 0 \quad \text{for} \quad 0 \leq k < d_j \tag{2.3.8}$$

More precisely, we can use the derivatives of Equation (2.3.7) to write

$$\left(\sum_{i=0}^{n} p_i \delta p_i B_i^n \right)^{(k)} (\alpha_j) = -P^{(k)}(\alpha_j)\,. \tag{2.3.9}$$

In order to find the derivatives in (2.3.9), we can use the differentiation matrix $\mathbf{D_B}$ in the Bernstein basis which is introduced in [29]. We note that it is a sparse matrix with only 3 nonzero elements in each column [29, Section 1.4.3]. So for each root α_i, we get d_i equations of the type (2.3.9). This gives us a linear system in the δp_i's. Solving the above linear system using the Singular Value Decomposition (`SVD`) one gets the desired solution.

Algorithm 3 gives a numerical solution to the problem. For an analytic solution for one single root see [22], [23], [12] and [11].

Algorithm 3 `Approximate-Polynomial`(P, L)

Input: P : list of coefficients of a polynomial of degree n in a Bernstein basis
L : list of pairs of roots with their multiplicities.
Output: $\tilde{P}$ such that for any $(\alpha, d) \in L$, $(x-\alpha)^d | \tilde{P}$.

1: $Sys \leftarrow EmptyList$
2: $D_B \leftarrow$ Differentiation matrix in the Bernstein basis of size $n+1$
3: $X \leftarrow \begin{bmatrix} x_1 & \dots & x_{n+1} \end{bmatrix}^t$
4: $T \leftarrow \texttt{EntrywiseProduct}(\texttt{Vector}(P), X)$
5: for $(\alpha, d) \in L$ do
  $A \leftarrow I_{n+1}$
  for i from 0 to $d-1$ do
    $A \leftarrow D_B \cdot A$
    $eq \leftarrow \texttt{DeCasteljau}(A \cdot T, \alpha) = -\texttt{DeCasteljau}(A \cdot \texttt{Vector}(P), \alpha)$
    $\texttt{append}(Sys, eq)$
6: Solve Sys using `SVD` to get a solution with minimal norm (such as 2.2.4), and return the result.

Although Algorithm 3 is written for one polynomial, in practice we apply it to both P and Q separately with the appropriate lists of roots with their multiplicities to get $\tilde{P}$ and $\tilde{Q}$.

2.4 Computing Approximate GCD

Assume the polynomials $P(x) = \sum_{i=0}^{n} a_i B_i^n(x)$ and $Q(x) = \sum_{i=0}^{m} b_i B_i^m(x)$ are given by their lists of coefficients and suppose $\alpha \geq 0$ and $\sigma > 0$ are given. Our goal here is to compute an approximate `GCD` of P and Q with respect to the given σ. Following Pan [21] as mentioned earlier, the idea behind our algorithm is to match the close roots of P and Q and then based on this matching find approximate polynomials $\tilde{P}$ and $\tilde{Q}$ such that their `GCD` is easy to compute. The parameter σ is our main tool for constructing the approximate polynomials. More precisely, $\tilde{P}$ and $\tilde{Q}$ will be constructed such that their roots are respectively approximations of roots of P and Q with σ as their error bound. In other words, for any root x_0 of P, $\tilde{P}$(similarly for Q) has a root $\tilde{x}_0$ such that $|x_0 - \tilde{x}_0| \leq \sigma$.

For computing approximate `GCD` we apply graph theory techniques. In fact the parameter σ helps us to define a bipartite graph as well, which is used to construct the approximate `GCD` before finding $\tilde{P}$ and $\tilde{Q}$.

We can compute an approximate `GCD` of the pair P and Q, which we denote by $\text{agcd}_\rho^\sigma(P(x), Q(x))$, in the following 5 steps.

Step 1. finding the roots: Apply the method of Section 2.2.1 to get $X = [x_1, x_2, \dots, x_n]$, the set of all roots of P and $Y = [y_1, y_2, \dots, y_m]$, the set of all roots of Q.

Step 2. forming the graph of roots $G_{P,Q}$: With the sets X and Y we form a bipartite graph, G, similar to [21] which depends on parameter σ in the following way:
If $|x_i - y_j| \leq 2\sigma$ for $i = 1, \dots, n$ and $j = 1, \dots, m$, then we can store that pair of x_i and y_j.

Step 3. find a maximum matching in $G_{P,Q}$: Apply the Hopcroft-Karp algorithm [13] to get a maximum matching $\{(x_{i_1}, y_{j_1}), \dots, (x_{i_r}, y_{j_r})\}$ where $1 \leq k \leq r$, $i_k \in \{1, \dots, n\}$ and $j_k \in \{1, \dots, m\}$.

Step 4. forming the approximate GCD:

$$\operatorname{agcd}_{\rho}^{\sigma}(P(x), Q(x)) = \prod_{s=1}^{r}(x - z_s)^{t_s} \tag{2.4.1}$$

where $z_s = \frac{1}{2}(x_{i_s} + y_{j_s})$ and t_s is the minimum of multiplicities of x_{i_s} and y_{j_s} for $1 \leq s \leq r$.

Step 5. finding approximate polynomials $\tilde{P}(x)$ and $\tilde{Q}(x)$: Apply Algorithm 3 with $\{z_1, \ldots, z_r, x_{r+1}, \ldots, x_n\}$ for $P(x)$ and $\{z_1, \ldots, z_r, y_{r+1}, \ldots, y_m\}$ for $Q(x)$.

For steps 2 and 3 one can use the tools provided in Section 2.2.3. We also note that the output of the above algorithm is directly related to the parameter σ and an inappropriate σ may result in an unexpected result.

2.5 Numerical Results

In this section we show small examples of the effectiveness of our algorithm (using an implementation in Maple) with two low degree polynomials in a Bernstein basis, given by their list of coefficients:

$$P := [5.887134, 1.341879, 0.080590, 0.000769, -0.000086]$$

and

$$Q := [-17.88416, -9.503893, -4.226960, -1.05336]$$

defined in Maple using Digits := 30 (we have presented the coefficients with fewer than 30 digits for readability). So $P(x)$ and $Q(x)$ are seen to be

$$\begin{aligned} P(x) :=& 5.887134\,(1-x)^4 + 5.367516\,x\,(1-x)^3 \\ &+ 0.483544\,x^2\,(1-x)^2 + 0.003076\,x^3\,(1-x) \\ &- 0.000086\,x^4 \end{aligned}$$

and

$$\begin{aligned} Q(x) :=& -17.88416\,(1-x)^3 - 28.51168\,x\,(1-x)^2 \\ &- 12.68088\,x^2\,(1-x) - 1.05336\,x^3 \end{aligned}$$

Moreover, the following computations is done using parameter $\sigma = 0.7$, and unweighted norm-2 as a simple example of Equation (2.2.4), with $r = 2$ and $\alpha = (1, \ldots, 1)$.

Using Theorem 2.2, the roots of P are, printed to two decimals for brevity,

$$[\ 5.3 + 0.0\,i,\quad 1.09 + 0.0\,i,\quad 0.99 + 0.0\,i,\quad 1.02 + 0.0\,i\ ]$$

This in turn is passed to `ClusterRoots` (Algorithm 1) to get

$$P_{\texttt{ClusterRoots}} := [[1.036 + 0.0\,i, 3], [5.3 + 0.0\,i, 1]]$$

where 3 and 1 are the multiplicities of the corresponding roots.

Similarly for Q we have:

$$[\ 1.12 + 0.0\,i,\quad 4.99 + 0.0\,i,\quad 3.19 + 0.0\,i\ ]$$

$\max_{\deg}\{P,Q\}$	$\deg(agcd^{\sigma}_{\rho}(P,Q))$	$\parallel P-\tilde{P} \parallel_2$	$\rho(P,\tilde{P})$	$\parallel Q-\tilde{Q} \parallel_2$	$\rho(Q,\tilde{Q})$
2	1	0.00473	0.11619	0.01199	0.05820
4	3	1.08900	1.04012	0.15880	0.15761
6	2	0.80923	0.75634	0.21062	0.31073
7	2	0.02573	0.04832	0.12336	0.02672
10	5	0.165979	0.22737	0.71190	0.64593

TABLE 2.1: Distance comparison of outputs and inputs of our approximate GCD algorithm on randomly chosen inputs.

which leads to

$$Q_{\texttt{ClusterRoots}} := [[3.19+0.0\,i,1],[4.99+0.0\,i,1],[1.12+0.0\,i,1]]$$

Again the 1's are the multiplicities of the corresponding roots.

Applying the implemented maximum matching algorithm in Maple (see Section 2.2.3), a maximum matching for the clustered sets of roots is

$$T_{\texttt{MaximumMatching}} :=[[\{4.99,5.30\},1],[\{1.03,1.12\},1]]$$

This clearly implies we can define (see Step 4 of our algorithm in Section 2.4)

$$\text{agcd}^{0.7}_{\rho}(P,Q) :=(x-5.145)(x-1.078)$$

Now the last step of our algorithm is to compute the approximate polynomials having these roots, namely $\tilde{P}$ and $\tilde{Q}$. This is done using Algorithm 3 which gives

$$\tilde{P} :=[6.204827,1.381210,0.071293,0.000777,-0.000086]$$

and

$$\tilde{Q} :=[-17.202067,-10.003156,-4.698063,-0.872077]$$

Note that

$$\parallel P-\tilde{P} \parallel_{\alpha,2} \approx 0.32 \leq 0.7 \text{ and } \parallel Q-\tilde{Q} \parallel_{\alpha,2} \approx 0.68 \leq 0.7$$

We remark that in the above computations we used the built-in function `LeastSquares` in Maple to solve the linear system to get $\tilde{P}$ and $\tilde{Q}$, instead of using the SVD ourselves. This equivalent method returns a solution to the system which is minimal according to norm-2. This can be replaced with any other solver which uses `SVD` to get a minimal solution with the desired norm.

As the last part of experiments we have tested our algorithm on several random inputs of two polynomials of various degrees. The resulting polynomials $\tilde{P}$ and $\tilde{Q}$ are compared to P and Q with respect to 2-norm (as a simple example of our weighted norm) and the root semi-metric which is defined in Section 2.2.3. Some of the results are displayed in Table 2.1.

2.6 Concluding remarks

In this paper we have explored the computation of approximate GCD of polynomials given in a Bernstein basis, by using a method similar to that of Victor Y. Pan [21]. We first use the companion pencil of Jónsson to find the roots; we cluster the roots as Zeng does to find the so-called pejorative manifold. We then algorithmically match the clustered roots in an attempt to find agcd_ρ^σ where ρ is the *root distance semi-metric*. We believe that this will give a reasonable solution in the Bernstein coefficient metric; in future work we hope to present analytical results connecting the two.

Bibliography

[1] B. Beckermann and G. Labahn. When are two numerical polynomials relatively prime? *Journal of Symbolic Computation*, 26(6):677 – 689, 1998.

[2] B. Beckermann, G. Labahn, and Ana C. Matos. On rational functions without Froissart doublets. *Numerische Mathematik*, 138(3):615–633, Mar 2018.

[3] Dario A. Bini and Ana Marco. Computing curve intersection by means of simultaneous iterations. *Numerical Algorithms*, 43(2):151–175, 2006.

[4] J. A. Bondy and U. S. R. Murty. Graph Theory (Graduate Texts in Mathematics 244). *Springer, New York*, 2008.

[5] Brad Botting, Mark Giesbrecht, and John May. Using Riemannian SVD for problems in approximate algebra. In *Proceedings of the International Workshop of Symbolic-Numeric Computation, 2005, pp. 209–219.*

[6] Robert M. Corless and Nicolas Fillion. A graduate introduction to numerical methods: from the viewpoint of backward error analysis. *Springer*, 2013.

[7] Robert M. Corless, Patrizia M. Gianni, Barry M. Trager, and Stephen M. Watt. The singular value decomposition for polynomial systems. In *Proceedings of the 1995 International Symposium on Symbolic and Algebraic Computation*, ISSAC '95, pages 195–207, New York, NY, USA, 1995. ACM.

[8] Rida T. Farouki and T. N. T. Goodman. On the optimal stability of the Bernstein basis. *Math. Comp*, 65:1553–1566, 1996.

[9] Rida T. Farouki and V. T. Rajan. On the numerical condition of polynomials in Bernstein form. *Computer Aided Geometric Design*, 4(3):191–216, 1987.

[10] Rida T. Farouki and V. T. Rajan. Algorithms for polynomials in Bernstein form. *Computer Aided Geometric Design*, 5(1):1 – 26, 1988.

[11] Markus A. Hitz, Erich Kaltofen, and Joseph E. Flaherty. Efficient algorithms for computing the nearest polynomial with constrained roots. *Rensselaer Polytechnic Institute, Troy, NY*, 1998.

[12] Markus A. Hitz, Erich Kaltofen, and Y. N. Lakshman. Efficient algorithms for computing the nearest polynomial with a real root and related problems. In *Proceedings of the 1999 international symposium on Symbolic and algebraic computation*, pages 205–212. ACM, 1999.

[13] John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matching in bipartite graphs. *SIAM J. Comput*, 2:225–231, 1973.

[14] Guðbjörn F. Jónsson. Eigenvalue methods for accurate solution of polynomial equations. *PhD dissertation, Center for Applied Mathematics, Cornell University, Ithaca, NY*, 2001.

[15] Guðbjörn F. Jónsson and Stephen Vavasis. Solving polynomials with small leading coefficients. *SIAM Journal on Matrix Analysis and Applications*, 26(2):400–414, 2004.

[16] Erich Kaltofen, Zhengfeng Yang, and Lihong Zhi. Structured low rank approximation of a Sylvester matrix. *In Symbolic-numeric computation*, pages 69–83. Springer, 2007.

[17] N. K. Karmarkar and Y. N. Lakshman. Approximate polynomial greatest common divisors and nearest singular polynomials. In *Proceedings of the 1996 International Symposium on Symbolic and Algebraic Computation*, ISSAC '96, pages 35–39, New York, NY, USA, 1996, ACM.

[18] N. K. Karmarkar and Y. N. Lakshman. On approximate gcds of univariate polynomials. *Journal of Symbolic Computation*, 26(6):653–666, 1998.

[19] D. Steven Mackey and Vasilije Perović. Linearizations of matrix polynomials in Bernstein bases. *Linear Algebra and its Applications*, 501:162–197, 2016.

[20] Y. Nakatsukasa, O. Séte„ and L. Trefethen. The AAA algorithm for rational approximation. *SIAM Journal on Scientific Computing*, 40(3):A1494–A1522, 2018.

[21] Victor Y. Pan. Numerical computation of a polynomial gcd and extensions. *Information and computation*, 167:71–85, 2001.

[22] Nargol Rezvani and Robert M. Corless. The nearest polynomial with a given zero, revisited. *ACM SIGSAM Bulletin*, 39(3):73–79, 2005.

[23] Hans J. Stetter. The nearest polynomial with a given zero, and similar problems. *ACM SIGSAM Bulletin*, 33(4):2–4, 1999.

[24] D. B. West. Introduction to graph theory. *Prentice Hall, Inc., Upper Saddle River, NJ*, 1996.

[25] Joab R. Winkler and N. Yang. Resultant matrices and the computation of the degree of an approximate greatest common divisor of two inexact Bernstein basis polynomials. *Computer Aided Geometric Design*, 30(4): 410–429, 2013, Elsevier.

[26] Zhonggang Zeng. The numerical greatest common divisor of univariate polynomials. *Randomization, relaxation, and complexity in polynomial equation solving* **556** (2011), 187–217.

[27] Zhonggang Zeng. The approximate gcd of inexact polynomials. part I: a univariate algorithm. *Proceedings of the 2004 international symposium on Symbolic and algebraic computation, ACM*, (2004), 320–327.

[28] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein. Introduction to algorithms second edition. *The Knuth-Morris-Pratt Algorithm*, 2001.

[29] A. Amiraslani and Robert M. Corless and M. Gunasingam. Differentiation matrices for univariate polynomials. *Numerical Algorithms, Springer*, 1–31, 2018.

Chapter 3

Approximate GCD in Lagrange bases

3.1 Introduction

The Euclidean algorithm or its variant, the extended Euclidean algorithm, is one of the most well-known and useful algorithms in symbolic computation. One important role of this rational algorithm for finding greatest common divisors of polynomials in computer algebra is to identify, in the case where all polynomials are exact, polynomials that have multiple roots, and indeed to compute a square-free factoring of a polynomial by starting with the computation of the GCD of the polynomial and its derivative. Although computing the GCD of two polynomials is straightforward in exact arithmetic and although much effort has been expended to find more efficient algorithms than the Euclidean algorithm such as using subresultants, over $\mathbb{R}$ or $\mathbb{C}$ the GCD problem is ill-posed. More precisely, it is an easy exercise to provide a pair of univariate polynomials with real or complex coefficients where a small perturbation (modelling noise or ignorance of the data in the original problem) changes a non-trivial GCD into a constant GCD.

In order to deal with ill-posedness of the GCD problem, one can look at the approximate GCD problem. For a given pair P and Q of univariate polynomials over reals, find another pair $\tilde{P}$ and $\tilde{Q}$ such that they are respectively "close" to P and Q, with a nontrivial GCD. and their exact GCD, is called an *approximate GCD* of P and Q. Indeed the closeness notion here needs clarification. For now let us just say a pseudometric on the space of polynomials of an specific degree gives us a tool for this closeness (see Section 3.2).

There is a vast literature on the approximate GCD problem most of which are devoted to study this problem for polynomials given in power basis. We invite the reader to take a look at [37], since the main ideas of this paper are borrowed from there. There are many interesting results in approximate GCD including but not limited to [8–10, 27, 31–33, 36, 42, 43] for approximate GCD problem in power basis.

Although the power basis is the most well understood and common basis to represent polynomials, it is not the only one. For various purposes, sometimes we would rather to have a polynomial in another basis such as a Bernstein basis. One of these cases happens when one wants to find intersection of Bézier curves which in turn has applications to Computer Aided Geometric Design. A theoretical solution is to convert the given pair of polynomials in a Bernstein basis into the power basis. In practice it is not efficient since the conversion is unstable and higher degrees cause more instability [7]. In [19] an algorithm for finding an approximate GCD in a Bernstein basis is given which also follows the ideas of [37].

In many applications, one needs to do computations with polynomials given in a Lagrange basis. More precisely, these polynomials are given by their values at certain points. For a similar reason as in Bernstein bases, conversion is not a good option here as well. A further reason is that Lagrange bases themselves are often *well*-conditioned, and working with them directly preserves numerical stability [21]. See also [12]. Direct computations in Lagrange bases are investigated in [1–4, 15, 38, 39]. Applications of these ideas and similar are investigated in, for instance, [5, 11, 22, 25]. In this paper we present an algorithm which follows the same ideas as in [37] and [19] and solves the approximate GCD problem in Lagrange bases.

Our main working assumption is that the given data is *near* to data specifying a pair of polynomials with common roots. We allow the problem solver or user to specify a tolerance to say roughly just *how* near, but the method of this paper cannot work if the given data is $O(1)$ away from polynomials with multiple roots. To see this, one may zero out both polynomials with an $O(1)$ change in the polynomial values, or otherwise trivially make them identical.

Our algorithm works with the sets of roots of a given pair of polynomials. Assume P and Q are given with $\deg(P) = n$ and $\deg(Q) = m$, $\sigma > 0$ and ρ_t is a metric (or pseudometric) on the space of polynomials of degree t. We will remind the reader what a metric is, and give the definition of a pseudometric, in Section 3.2.

We first compute all roots of P and Q using the companion pencil method introduced in [15]. The second step is to cluster the roots of each polynomial, i.e. find close roots and replace them by one single root with an appropriate multiplicity. The next step is to form a bipartite weighted graph with sets of vertices which are the clustered roots of P and Q. For r and s with $P(r) = Q(s) = 0$, $\{r, s\}$ is an edge with weight

$$W(\{r, s\}) = \min\{\text{multiplicity}_P(r), \text{multiplicity}_Q(s)\}.$$

Then we use a *Maximum Weighted Matching (MWM)* algorithm to find a maximum matching, say M, and using M we form approximate GCD. The last step is to construct $\tilde{P}$ and $\tilde{Q}$ where $\rho_n(P, \tilde{P}), \rho_m(Q, \tilde{Q}) \leq \sigma$. This step is simpler in a Lagrange basis when it is compared to a Bernstein basis or the power basis provided that ρ_t be a well behaved metric (pseudometric).

This paper is organized in the following way. Section 3.2 is devoted to provide the necessary formal definitions for approximate GCD. A method for computing roots of polynomials in Lagrange basis is provided in the third section. Section 3.4 introduces a divide and conquer algorithm for clustering roots of a polynomial, together with an alternative set of heuristics that uses symmetry. The maximum weight matching problem is discussed in Section 3.5. Section 3.6 contains details of computing an approximate GCD and cofactors of polynomials $\tilde{P}$ and $\tilde{Q}$. Finally we provide numerical experiments using a Maple implementation of our algorithm.

3.2 Definitions

In this brief section we provide necessary formal definitions which lead to a careful definition of an approximate GCD. The most crucial component of the notion of an approximate GCD is "closeness". Hence, we need to formally present a metric on the set of degree n polynomials. Although a metric specifies the notion of closeness completely, a *pseudometric* does the job as well.

Definition 3.1. [34] A metric for a set X is a function d on the cartesian product $X \times X$ to the non-negative reals such that for all points x, y, and z of X,

(a) $d(x, y) = d(y, x)$,

(b) (triangular inequality) $d(x,y)+d(y,z)\geq d(x,z)$,

(c) $d(x,y)=0$ if $x=y$, and

(d) if $d(x,y)=0$, then $x=y$.

A pseudometric on a set X is a non-negative real valued function ϕ defined on $X\times X$ and satisfies all the properties of a metric except for possibly for property (d) which is called the distinguishability property. On the other hand, since a polynomial can be presented with different data in Lagrange basis, regular well-known metrics are not well-defined.

Example 3.1. *Assume $T_3(x)$, the third Chebyshev polynomial of the first kind, is given with two different representations in 2 Lagrange bases:*

$$L_1=\begin{bmatrix}-1 & -\frac{1}{2} & \frac{1}{2} & 1\\ -1 & 1 & -1 & 1\end{bmatrix},\quad L_2=\begin{bmatrix}-2 & -\frac{1}{3} & \frac{1}{3} & 3\\ -26 & \frac{23}{27} & -\frac{23}{27} & 99\end{bmatrix}.$$

Now, suppose we want to use 2-norm (of matrices). Using Maple we get

$$\parallel L_1-\mathbf{0}\parallel_2=\frac{3\sqrt{2}}{2},$$

and

$$\parallel L_2-\mathbf{0}\parallel_2=\sqrt{\frac{3824215}{729}+\frac{5\sqrt{584620384633}}{729}}.$$

This shows that the 2-norm is not well defined in Lagrange bases. Moreover, one can get $\parallel L_1-L_2\parallel=\dfrac{3329}{27}$, which shows that the property (c) does not hold.

In order to give a well-defined metric (or pseudometric) we have to consider invariants of polynomials such as their roots.

In [19] we used the so-called *root semi-metric* to solve the approximate GCD problem in Bernstein bases. In this work we use the same idea except that we consider it only on polynomials with same degree. This leads to a well-defined pseudometric which we call it the *root-pseudometric*.

Assume ρ is metric on $\mathbb{C}^n$, S_n is the group of permutations of $\{1,\ldots,n\}$. Moreover, for $f\in\mathbb{R}[x]_n$ (set of univariate polynomials of degree n with real coefficients) $\{f_1,\ldots,f_n\}$ is the set of roots of f. Let $R_f=(f_1,\ldots,f_n)$ and $\tau\in S_n$ acts on R_f by acting on its coordinates, i.e.

$$\tau(R_f)=\left(f_{\tau(1)},\ldots,f_{\tau(n)}\right).$$

The map

$$\begin{array}{rccl}\mathbf{d}_n:&\mathbb{R}[x]_n\times\mathbb{R}[x]_n&\longrightarrow&\mathbb{R}_{\geq 0}\\ &(f(x),g(x))&\longmapsto&\dfrac{1}{n}\min\limits_{\tau\in S_n}\left\{\rho\left(\tau(R_f),R_g\right)\right\}\end{array}\tag{3.2.1}$$

is a pseudometric on $\mathbb{R}[x]_n$. Let

$$Rep_{\mathbf{d}_n}(f,g)=\{\tau\in S_n:\mathbf{d}_n(f,g)=\frac{1}{n}\rho(\tau(R_f),R_g)\}.$$

In fact the identity element of S_n gives $\mathbf{d}_n(f,f)=0$ and

$$\tau\in Rep_{\mathbf{d}_n}(f,g)\Longrightarrow\tau^{-1}\in Rep_{\mathbf{d}_n}(g,f),$$

where τ^{-1} is the inverse of τ in S_n. Hence $\mathbf{d}_n(g,f) = \mathbf{d}_n(f,g)$. The last property of a pseudometric is the triangle inequality. For any fixed $h \in \mathbb{R}[x]_n$ assume $\lambda \in Rep_{\mathbf{d}_n}(f,g)$, $\tau \in Rep_{\mathbf{d}_n}(f,h)$, $\gamma \in Rep_{\mathbf{d}_n}(g,h)$. Now if $\mathbf{d}_n(f,h) + \mathbf{d}_n(g,h) < \mathbf{d}_n(f,g)$ then

$$\rho(\tau(R_f), R_h) + \rho(\gamma(R_g), R_h) < \rho(\lambda(R_f), R_g),$$

since ρ is a metric on $\mathbb{C}^n$,

$$\rho\left(\tau(R_f), \gamma(R_g)\right) \leq \rho(\tau(R_f), R_h) + \rho(\gamma(R_g), R_h) < \rho(\lambda(R_f), R_g).$$

This in particular shows that $\rho(\lambda(R_f), R_g)$ was not minimum which is a contradiction.

It is worth mentioning that the distinguishability property does not hold for $\mathbf{d}_n$. For a non-zero constant c, $\mathbf{d}_n(cf, f) = 0$.

Now that we have a metric in hand, we can formally define an approximate GCD for a pair of polynomials. A *pseudogcd* set for the pair P and Q is defined as

$$A_{\alpha,r,\sigma} = \{g(x) \mid \exists \tilde{P}, \tilde{Q} \text{ with } \mathbf{d}_{\deg(P)}(P - \tilde{P}) \leq \sigma, \\ \mathbf{d}_{\deg(Q)}(Q - \tilde{Q}) \leq \sigma \text{ and } g(x) = \gcd(\tilde{P}, \tilde{Q})\}. \quad (3.2.2)$$

Definition 3.2. An approximate GCD for P, Q which is denoted by $\operatorname{agcd}_\rho^\sigma(P,Q)$, is $G(x) \in A_{\alpha,r,\sigma}$ where $\deg(G) = \max_{g \in A_{\alpha,r,\sigma}} \deg(g(x))$, and $\mathbf{d}_{\deg(P)}(P - \tilde{P}), \mathbf{d}_{\deg(Q)}(Q - \tilde{Q})$ are minimal.

3.3 Finding roots of a polynomial in Lagrange basis

It is well known that solving polynomial equations, or finding the eigenvalues of matrix polynomials, can be done by converting to a generalized eigenvalue problem. In this section we want to find the roots of a given polynomial in Lagrange basis. There are various methods for solving this problem. For example in [28] the author uses a method of Smith [40] that has not been generalized to matrix polynomials. However, in this work we prefer to use the generalized eigenvalue problem to find the roots of a certain polynomial.

We recall the method introduced in [15] for finding zeros of a given polynomial in Lagrange basis. Consider $p_0, p_1, \ldots, p_n$ which are the values of a polynomial P at $x = x_0$, $x = x_1, \cdots$, and x_n. Then *the generalized companion matrix* of the polynomial given by its value is

$$\mathbf{C}_0 = \begin{bmatrix} 0 & -p_0 & -p_1 & \cdots & -p_n \\ \ell_0 & x_0 & & & \\ \ell_1 & & \ddots & & \vdots \\ \vdots & & & & \\ \ell_n & & \cdots & & x_n \end{bmatrix} \quad (3.3.1)$$

where $\ell_k = 1/\prod_{j \neq k}(x_k - x_j)$ are the (scalar) normalization factors of the Lagrange polynomials $L_k(x) = \ell_k \prod_{j \neq k}(x - x_j)$ and

$$\mathbf{C}_1 = \begin{bmatrix} 0 & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 \end{bmatrix}. \quad (3.3.2)$$

Then we have

Theorem 3.3. *[15] With the above notation and assumptions*

$$\det(x\boldsymbol{C}_1 - \boldsymbol{C}_0) = P(x).$$

The following example shows how the above theorem can be used in practice.

Example 3.2. *Assume polynomial $f(x)$ is given by:*

$$f_x = [4.1, -2.2, 1.22, 5.5, 3.23, 8.1, 9.2]$$

and

$$f_y = [-2306.90, -9.41, -4827.64, 182.10, -4306.04, 3856.85, 28326.04]$$

where f_x contains nodes and f_y provides values of f on the nodes. The corresponding companion pencil to f is

$$C_0 = \begin{bmatrix} 0 & 2306.90 & 9.41 & 4827.64 & -182.10 & 4306.04 & -3856.85 & -28326.04 \\ -0.002 & 4.1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.000009 & 0 & -2.2 & 0 & 0 & 0 & 0 & 0 \\ -0.0002 & 0 & 0 & 1.22 & 0 & 0 & 0 & 0 \\ 0.0009 & 0 & 0 & 0 & 5.5 & 0 & 0 & 0 \\ 0.0015 & 0 & 0 & 0 & 0 & 3.23 & 0 & 0 \\ -0.0002 & 0 & 0 & 0 & 0 & 0 & 8.1 & 0 \\ 0.00008 & 0 & 0 & 0 & 0 & 0 & 0 & 9.2 \end{bmatrix}$$

$$C_1 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Notice that the first row, which contains the values of the polynomial, seems to have a somewhat large norm. Likewise notice that the first column, which contains the barycentric weights, quantities that depend on the nodes and not the values, seems to have a somewhat small norm. In some cases, these poor scalings can lead to numerical difficulty. However, the first row can be multiplied by any nonzero constant whatever without affecting the eigenvalues; likewise the first column. This can be proved by examining the second barycentric form [16]. Thus we may choose to scale this matrix so its first row and first column have, say, norm 1 in any reasonable norm. In what follows we will not comment on such scalings unless we have to make them for numerical reasons. In this present example we do not scale the matrix, but merely call the numerical software on the "raw" matrix pair.

The Maple command `Eigenvalues` *returns*

$$\begin{bmatrix} Float(\infty) + 0.0\,i \\ Float(\infty) + 0.0\,i \\ -2.49 + 0.0\,i \\ -2.09 + 0.0\,i \\ -1.70 + 0.0\,i \\ 7.10 + 0.0\,i \\ 6.79 + 0.0\,i \\ 5.30 + 0.0\,i \end{bmatrix}$$

One can verify

$$f(-2.49 + 0.0\,i) = 0$$
$$f(-2.09 + 0.0\,i) = 0$$
$$f(-1.70 + 0.0\,i) = 0$$
$$f(7.10 + 0.0\,i) = 0$$
$$f(6.79 + 0.0\,i) = 0$$
$$f(5.30 + 0.0\,i) = 0$$

Note that in examples through this work, we set $Digits := 30$ *(in Maple). However, for readability, we have showed only a few digits.*

3.4 Clustering the roots

As we discussed in the first section, our algorithm gets two polynomials as input. In the previous section we explained the method of which our algorithm is using to find roots of each polynomial. We note that our method is numerical and the computed roots are approximations. So it is natural to wonder if a root with a multiplicity more than 1 is approximated as different roots which are close. Although it is not so clear when this really has happened, we believe working with roots while merging close enough roots is a reasonable approach and likely to lead to good results. This corresponds to what Kahan terms projecting onto the "pejorative manifold". We call the process of identifying close roots as the same root with appropriate multiplicity, *root clustering*.

On the other hand, we can also execute this step in the algorithm after constructing an approximate `GCD`. However, to be consistent with the other results in the literature, we put clustering as the second step of our algorithm.

The problem of properly clustering complex roots of perturbed polynomials is hard, and depends strongly on the model used for the perturbation. In the papers [6, 29], for instance, we find the use of an *oracle*, which assumes that information about the *exact* or "underlying" polynomial is available at the request of the user (which might be a program) and in exchange for more computational cost. That model has been used in polynomial computation since the pioneering work of Schönhage and leads to significant algorithms of practical interest.

However, our model is different: as in [17] we assume that no oracle is available, and all we have is noisy data about the polynomial (in the Lagrange case, approximate values at well-specified nodes). In some sense this makes the problem actually impossible:

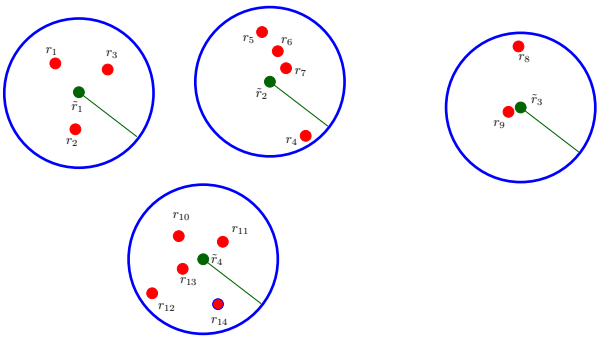

FIGURE 3.1: Root Clustering for $P(x) = \prod_{i=1}^{14}(x - r_i)$ is found as $P(x) = \prod_{i=1}^{4}(x - \tilde{r}_i)$. Here only distance is considered and symmetry is ignored.

there may be several pairs of polynomials nearby that have common roots. From the incomplete data that we have, however, we must recover the approximate GCD with our best estimate of the multiplicities. To do this, we resort to heuristics. We present one such algorithm, that uses symmetry and distance, and we also present a divide and conquer algorithm which follows a different idea. See Figure 3.1 for an example of a clustering without using symmetry.

3.4.1 Heuristics for clustering complex roots

Our first heuristic is the use of distance. When a polynomial with a multiple root is perturbed, say to $q(z)(z - r)^m + \Delta p(z)$, then the multiple root is, to first order, if $\Delta p(r) = s$ is small,

$$z \approx r + \left(-\frac{\Delta p(r)}{q(r)}\right)^{1/m} . \tag{3.4.1}$$

This leads to a cluster of m simple roots at a distance $(s/|q(r)|)^{1/m}$ away, and if s is "small" these will be equally-spaced about a near-circle with that radius. In our current work we do not estimate the size of $q(r)$ but rather use some modest constants or "fuzz factors" for this; in future work we will refine this by using the division algorithm of [1] to give an estimate of $q(r)$ itself.

The second heuristic is the use of the equal-angle property mentioned above. Indeed this seems to be quite a reliable indication that we have the correct multiplicity, although we need more experimentation to assert true confidence.

The third heuristic is to look through the list of approximate roots for clusters of higher multiplicity *first*. This supplies a bias towards higher multiplicity, or equivalently to projection onto the so-called *pejorative manifold*.

Access to these heuristics is possible only because we are looking at approximate roots as a method of finding approximate GCD. Such heuristics are likely to be less valuable for other approaches to approximate GCD in the Lagrange basis, e.g. via the Bézout matrix [5, 24, 39].

The heuristics are easily fooled for roots of *very* high multiplicity, so our routine has a default maximum order of multiplicity of 3. We have used it successfully on artificial examples of multiplicity as high as 5, however.

Let us exhibit our heuristics with a pseudorandom sample of $n = 20$ points in the square $[0, 1] \times [0, 1]$; we used Maple's `rand()` function twenty times, dividing its integer result by 10^{12}; half we took for x-values and the other half for y-values in $z = x + iy$.

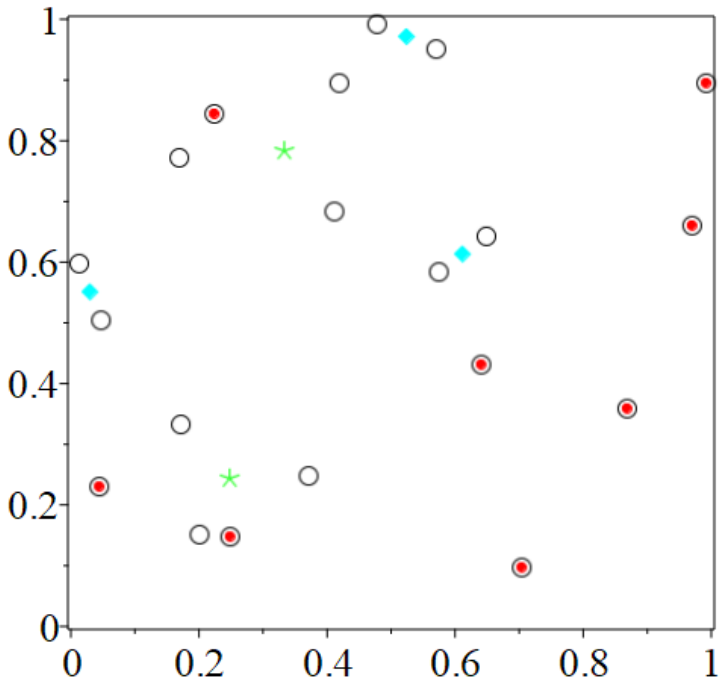

FIGURE 3.2: Original data plotted with open black circles. Clustered triple points plotted with green asterisks. Clustered double points plotted with cyan diamonds. Points left untouched by clustering plotted with solid red circles. Note that the triple point clusters each preferred a farther point to include than one that could have been chosen: evidently the heuristic for symmetry is working.

We clustered these points with a tolerance of $1/n^2$: one expects a few of n points in the unit square to be $O(1/n^2)$ close. The results are plotted in Figure 3.2.

Two sets of triple points were found. Each of them has the interesting feature that there is one unclustered point closer to the triple's centre than one of the ones chosen. We believe that the more symmetrical point was chosen, even though it was farther away.

Three sets of double points were found; from the graph, these choices seem very reasonable.

3.4.2 The effect of conditioning: Wilkinson's example

Wilkinson's famous example

$$\prod_{k=1}^{20}(x-k) = x^{20} - 210\,x^{19} + 20615\,x^{18} - 1256850\,x^{17} + 53327946\,x^{16} - \cdots + 20! \quad (3.4.2)$$

has, as is very well known, different condition numbers in the two different representations. See [20] for a detailed exposition. In that paper, as in [21], we also see that *random* Lagrange bases, where the nodes are chosen uniformly at random on the interval $[0, 20]$, are also surprisingly well-conditioned.

This allows us to demonstrate that the root pseudometric is connected to the coefficient vector metric via the condition number. Alternatively, it is connected by the notion of *pseudozeros*, the set of all roots of all polynomials "nearby" in the coefficient metric. We show by example that a small perturbation in the root pseudometric will, if the polynomial representation is well-conditioned, generate a *larger* perturbation in the coefficient vector; contrariwise, if the polynomial representation is ill-conditioned, then a small perturbation in the root pseudometric will make an even *smaller* perturbation in the coefficient vector. This is of course because we are using the relationship in the reverse way to usual.

For example, expanding Wilkinson's polynomial into the monomial basis produces a famously ill-conditioned polynomial representation. If we find the zeros numerically from that representation (say by its Frobenius companion matrix using 15 decimal Digits in Maple) we get the roots (trimming digits after the first inconvenient one just to save space)

$$\begin{aligned}&[0.999999999999, 2.0000000001, \cdots\\ &11.24, 11.64, 13.48 + 0.3917\,i, 13.48 - 0.3917\,i,\\ &\cdots 19.009, 19.999]\end{aligned}$$

where we see some of the computed roots are now complex. If we use our method to cluster all these roots, using a tolerance of 0.05, we get

$$\begin{aligned}&[[11.441, 2], [15.556, 2], [1.0, 1], [2.0, 1],\\ &[3.0, 1], [4.0, 1], [5.0, 1], [6.0, 1],\\ &[7.0005, 1], [7.9960, 1], [9.0206, 1], [9.9342, 1],\\ &[13.481 + 0.39171\,i, 1], [13.481 - 0.39171\,i, 1], [17.138, 1],\\ &[17.948, 1], [19.009, 1], [19.999 + 0.0\,i, 1]]\,.\end{aligned} \tag{3.4.3}$$

Exactly fitting a monic monomial basis polynomial to this (converted to exact rational) data, we find that the resulting coefficient vector differs from the original only by less than $2 \cdot 10^{-5}\|\vec{W}\|_\infty$, where $\vec{W} = [1, -210, \ldots, 20!]$ is the vector of monomial basis coefficients. That is, a root change of about 0.05 corresponds to a coefficient change twenty-five hundred times smaller—this is because the coefficients are ill-conditioned and sensitive.

In contrast, choose 21 nodes at random on $[0, 20]$ and use the product definition of the Wilkinson polynomial (which is *extremely* well-conditioned) to evaluate it on these nodes, to create a simulated problem to consider. We now have 21 node-value pairs, which gives us a Lagrange basis representation. Compute the companion matrix of [15] (available in Maple as `LinearALgebra[CompanionMatrix]`, see [30]) and find its eigenvalues, which will be the roots of the polynomial. When we did this, and applied our clustering heuristics with tolerance 0.05 to the results, we got the (somewhat surprising) answer

$$\begin{aligned}&[[16.823, 2], [4.3500, 2], [14.837, 2], [9.3079, 3],\\ &[7.6965, 3], [19.928, 1], [18.898, 1],\\ &[2.1301, 1], [15.416, 1], [13.880, 1],\\ &[12.317, 1], [6.6043, 1], [8.8976, 1]]\,.\end{aligned} \tag{3.4.4}$$

This has two *triple* roots, which we were not expecting, as well as three double roots. When we convert this to the very well-conditioned Hermite representation $K(x-r_1)^2(x-$

$r_2)^2(x-r_3)^2(x-r_4)^3(x-r_5)^3\cdots(x-r_{13})$ and choose K as best we can[1], we see that the values of this polynomial on the original nodes form a vector that is *at least* 47% different to the original values of the polynomial. That is, changes in the roots on the order of 0.05 can only happen if the coefficients change by around a factor of one-half: this is not a small change.

This is because the Lagrange basis on those random nodes uniformly chosen in the region where the roots actually are is actually an extremely *well*-conditioned representation. Small changes in those values will produce only very tiny changes in the roots. Even a modest change in the roots requires a significant change to the values (which are the representation of the polynomials on those nodes).

We will discuss a formal connection of the coefficient vector perturbation with the root pseudometric in Section 3.8.

3.4.3 A Divide and Conquer Clustering Algorithm

In this subsection we present a deterministic clustering algorithm which only considers a tolerance directly to cluster a set of given points in the complex plain. Our algorithm is a slight modification of the divide and conquer, *Closest Pair Algorithm* [23, P. 958-960].

Assume Q is an array of roots of a polynomial. We apply a sorting algorithm to $Q=(r_1,\ldots,r_n)$ and get the array of roots sorted with respect to their real parts. This can be done in $O(n\log n)$. We also add multiplicities of r_i's to them by replacing each of them by $(r_i, mult(r_i))$.

Our algorithm has 3 steps:

Step 1: Split the points into two almost equal parts, Q_L and Q_R by considering the line L, defined as $x=\Re(Q[m][1])$ where $m=\lfloor\frac{\ell+r}{2}\rfloor$, ℓ and r are left and right boundaries in recursive calls. Recursively call the algorithm on Q_L and Q_R then merge the sets (consider imaginary parts for merging).

Step 2: Once Q_L and Q_R are clustered, form an strip containing the points from them which have distance less than σ to middle line (see Figure 3.3).

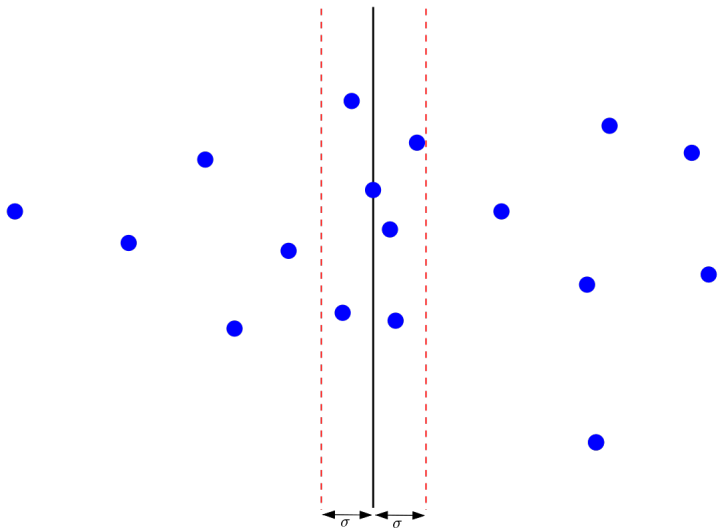

FIGURE 3.3: Middle strip for merging the left and right results in clustering algorithm

[1]We tried several things, including a matching of the average absolute value, and a least-squares fit.

Step 3: Check the strip for close pairs and merge them. Note that for this step (u, d_u) and (v, d_v) will be merged as

$$\left(\frac{d_u \cdot u + d_v \cdot v}{d_u + d_v}\right)^{d_u+d_v},$$

if $|u - v| \leq \sigma$.

Algorithm 4 `ClusterRoots`(Q, ℓ, r, σ)

Input: A sorted list of roots with their multiplicities, Q.
ℓ is the left boundary (initially $\ell \leftarrow 1$)
r is the right boundary (initially $r \leftarrow$ `Size`(Q)).
σ is the tolerance for clustering.
Output: Clustered roots. $S \leftarrow Q$
if $\ell = r$ **then**
$S \leftarrow [Q[l]]$
else
$m \leftarrow \lfloor(\ell + r)/2\rfloor$
$S_L \leftarrow$ `ClusterRoots`(Q, ℓ, m, σ)
$S_R \leftarrow$ `ClusterRoots`$(Q, m + 1, r, \sigma)$
$S \leftarrow$ `Merge`$_{\Im}(Q_L, Q_R)$
$R \leftarrow$ `Select`$(S, \Re(S[mid][1]), \sigma)$
$i \leftarrow$ `Search`$(S_L, R[first])$
$j \leftarrow$ `Search`$(S_R, R[last])$
$T \leftarrow$ `CheckStrip`(R, σ)
$S \leftarrow$ `Merge`$(S_L[1..i-1], R, S_R[j+1..last])$
return(S)

An argument which shows that the cost of merging is $O(n)$ is presented in [23]. This means the running time is recursively given by $T(n) = 2T(\frac{n}{2}) + O(n)$ and using the Master theorem [23] we can resolve it to $T(n) \in O(n \log n)$. Hence the total cost is $O(n \log n)$.

Example 3.3. *We work with the roots computed in Example 3.2. We need to add multiplicities of roots to the list so that it is compatible with our clustering algorithm. Adding the multiplicities, we get:*

$$[[-2.499, 1], [-2.099, 1], [-1.700, 1], [5.300, 1], [6.799, 1], [7.100, 1]]$$

The output of `ClusterRoots` *(implemented in Maple) for* $\sigma = 0.5$ *is*

$$[[-2.299, 2], [-1.700, 1], [5.300, 1], [6.950, 2]].$$

Neither of the above algorithms for clustering roots is perfect. The divide and conquer algorithm does not give the highest degrees after clustering. As an example of its behavior, for the input

$$[[1, 1], [1.5, 1], [2, 1]],$$

and $\sigma = 0.5$ it returns

$$[[1.25, 2], [2, 1]],$$

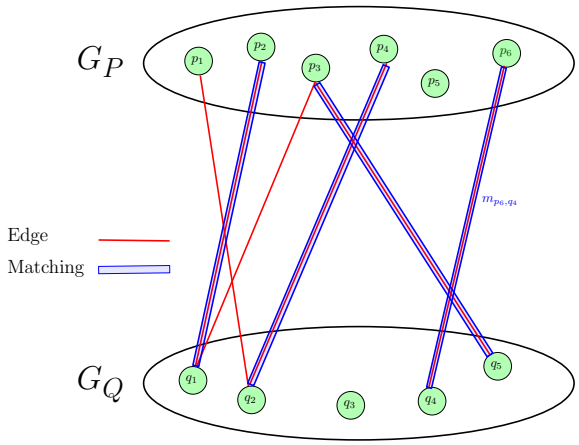

FIGURE 3.4: m_{p_6,q_4} is the minimum of multiplicities of p_6 and q_4.

while we know that $[1.5, 3]$ would be a better answer according to multiplicity. Moreover `ClusterRoots` considers the given roots as points in the complex plane without considering their properties as approximate roots of polynomials.

3.5 Maximum Weight Matching problem and approximate GCD

In order to find the GCD of two polynomials, we find their common roots. In other words, we look at their roots and find matches between them. Since it is an exact GCD this matching means equality. In the approximate GCD case, matching means close enough with respect to a metric.

The above approach in looking at approximate GCD suggests us to use matching algorithms from graph theory. In this brief section we explain the necessary concepts and an algorithm from graph theory.

Assume $G = (V, E, W)$ is a weighted graph. A subset $M \subseteq E$, is called a matching if no two edges in M share a common vertex. A maximum cardinality matching (MCM) is a matching of the maximum size as a set.

The weight of a matching (in a weighted graph) is defined as

$$W_M = \sum_{e \in M} W(e).$$

A matching M with the maximum weight is called a maximum weight matching (MWM).

Suppose $n = |V|$ and $m = |E|$. It is well known that MCM for a bipartite graph can be solved in time $O(m\sqrt{n})$ using the deterministic algorithm introduced by Hopcoft and Karp. A more general deterministic algorithm for a general graph was given by Micali and Vazirani in 1980 which runs in $O(m\sqrt{n})$. There are many more algorithms for solving the same problem, we just mention that a randomized algorithm to find a MCM in a general graph was given by Harvey in 2006 with complexity $O(n^\omega)$ where ω is the exponent of $n \times n$ matrix multiplication and [35] proves that $\omega \leq 2.373$.

In the case that the weights are integers, we define N to be the largest magnitude of a weight. An MWM for a bipartite graph can be found in $O(Nn^{\omega})$ using the randomized algorithm introduced by Sankowski in 2006.

Following [26], for $\delta \in [0,1]$ we call a matching δ-approximate matching if its weight is at least a factor δ of the optimal matching. Similarly a δ-MWM is the problem of finding a δ-approximate maximum weight matching. Duan and Pettie introduced an algorithm to solve δ-MWM for a graph with integer weights in $O(m(1-\delta)^{-1}\log N)$.

A much simpler approximate MWM algorithm is a greedy algorithm. It is well-known that a greedy algorithm solves the problem for a general graph for $\delta = \frac{1}{2}$ (see [26]). In our special case, the graph is bipartite and a very simple one. Our Maple implementation of this greedy algorithm gives very good results in practice. We present the pseudocode of the mentioned greedy algorithm here.

Algorithm 5 `GreedyMWM`(G)

Input: A graph G given with a list of edges and their weights
$[W(\{a,b\}),\{a,b\}]$ which is sorted w.r.t weights.
Output: A Matching $M \subseteq E_G$.
$M \leftarrow [\,]$
for g in G
if $g[2]$ has no intersection with (2nd component of) elements in M then
`append`(M,g)
return M

One can verify that the running time of the above algorithm is $O(m)$.

We complete this section by describing the construction of a bipartite graph corresponding to a given pair of polynomials. Here we use similar ideas which was used in [19] except that this time we use the algorithm by [26] to get a δ-MWM.

Now we can form a bipartite weighted graph using the roots of the given polynomials P and Q. Assume R_P and R_Q are sets of roots of P and Q respectively. Let $G^{\sigma}_{PQ} = (V_P, V_Q, E_{PQ}, W_{PQ})$ with

- $V_P = \texttt{ClusterRoots}(R_P, \sigma)$,
- $V_Q = \texttt{ClusterRoots}(R_Q, \sigma)$,
- $E^{\sigma}_{PQ} = \Big\{\{r,s\} :\ r \in V_P, s \in V_Q, |r-s| \leq \sigma\Big\}$,
- $W_{PQ}(\{r,s\}) = \min\{\text{multiplicity}_P(r), \text{multiplicity}_Q(s)\}$.

A simple argument shows that we can form the corresponding graph to a pair of polynomials by doing n^2 comparisons.

3.6 Computing Approximate Polynomials and GCD

Assume M is an MWM for G^{σ}_{PQ}. Each element in M is an edge of the graph with a corresponding weight. We define

$$G(x) = \prod_{\{a,b\}\in M} \left(x - \left(\frac{\text{multiplicity}(a)\cdot a + \text{multiplicity}(b)\cdot b}{\text{multiplicity}(a) + \text{multiplicity}(b)}\right)\right)^{W(\{a,b\})} \tag{3.6.1}$$

to be the approximate GCD corresponding to M.

Using $\mathbf{d}_t$ (Equation (3.2.1)) on the space of polynomials of degree t we can find the approximate polynomials with desired properties. Suppose G is the approximate GCD as in Equation (3.6.1). Suppose $r_1, \dots r_u$ are clustered roots of P with multiplicity d_i which do not appear in the corresponding matching to G and $r_{u+1}, \dots, r_\ell$ with multiplicities $d_{u+1}, \dots, d_{u+\ell}$ are roots of P which appear in the corresponding matching to G. Similarly let $s_1, \dots, s_v, s_{v+1}, \dots s_{v+\ell}$ be the roots of Q with multiplicity d'_i. Moreover suppose, $w_i = W(r_{u+i}, s_{v+i})$ for $1 \leq i \leq \ell$.

We define

$$\tilde{P} = G(x) \cdot \prod_{1 \leq i \leq u} (x - r_i)^{d_i} \cdot \prod_{u+1 \leq i \leq \ell} (x - r_i)^{d_i - w_i} \tag{3.6.2}$$

$$\tilde{Q} = G(x) \cdot \prod_{1 \leq i \leq v} (x - s_i)^{d'_i} \cdot \prod_{v+1 \leq i \leq \ell} (x - s_i)^{d'_i - w_i} \tag{3.6.3}$$

Now it is not hard to see that there exists permutation of roots such that $\mathbf{d}_n(P, \tilde{P}) \leq \sigma$ and $\mathbf{d}_m(Q, \tilde{Q}) \leq \sigma$.

Since in each step of our algorithm we have access to roots (and their multiplicities) of polynomials, presenting them in a Lagrange basis is straightforward.

3.7 Numerical Results

In this section we present a concrete example of computing an approximate GCD by applying our algorithm to a pair of polynomials. In order to do so, we use a Maple implementation of our algorithm.

Assume P and Q are given with the following data (in a Lagrange basis):

$$\begin{aligned}
P_x =& [4.586334585, 5.161255391, 2.323567403, 1.809094426,\\
& 1.471852626, 4.427838553, 2.275731771, 1.020544909]\\
P_y =& [787.1243900, 3285.933680, 0.01345240, 0.00001680,\\
& 0.00880350, 499.6500860, 0.01132600, 0.12249270]\\
Q_x =& [2.812852786, 1.746745227, 2.296707006, 2.359573808,\\
& 4.747053250, 1.640439652, 5.832623175]\\
Q_y =& [-0.0095256, 0.0171306, 0.2253058, 0.2359018,\\
& 426.4319036, 0.0041690, 3314.165173]
\end{aligned}$$

where $P_y[i]$'s are values of P on $P_x[i]$'s and $Q_y[i]$'s are values of Q on $Q_x[i]$'s. Since P_x is of length 8 and Q_x is of length 7, polynomials P and Q are respectively of degrees 7 and 6.

The first step of our algorithm is to compute the roots of P and Q. We do this by forming the companion pencils. Corresponding to P we get

$$C_{P,0} = \begin{bmatrix}
0 & -787.12 & -3285.93 & -0.013 & -0.000016 & -0.0088 & -499.65 & -0.011 & -0.12\\
-0.068 & 4.58 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0.0056 & 0 & 5.16 & 0 & 0 & 0 & 0 & 0 & 0\\
-2.70 & 0 & 0 & 2.32 & 0 & 0 & 0 & 0 & 0\\
-0.64 & 0 & 0 & 0 & 1.80 & 0 & 0 & 0 & 0\\
0.28 & 0 & 0 & 0 & 0 & 1.47 & 0 & 0 & 0\\
0.072 & 0 & 0 & 0 & 0 & 0 & 4.42 & 0 & 0\\
3.09 & 0 & 0 & 0 & 0 & 0 & 0 & 2.27 & 0\\
-0.034 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1.02
\end{bmatrix}$$

and for Q we get

$$C_{Q,0} = \begin{bmatrix} 0 & 0.0095 & -0.017 & -0.22 & -0.23 & -426.43 & -0.0041 & -3314.16 \\ 0.58 & 2.81 & 0 & 0 & 0 & 0 & 0 & 0 \\ -2.13 & 0 & 1.74 & 0 & 0 & 0 & 0 & 0 \\ 9.85 & 0 & 0 & 2.29 & 0 & 0 & 0 & 0 \\ -9.60 & 0 & 0 & 0 & 2.35 & 0 & 0 & 0 \\ -0.0087 & 0 & 0 & 0 & 0 & 4.74 & 0 & 0 \\ 1.30 & 0 & 0 & 0 & 0 & 0 & 1.64 & 0 \\ 0.0014 & 0 & 0 & 0 & 0 & 0 & 0 & 5.83 \end{bmatrix}$$

Note that $C_{P,1}$ and $C_{Q,1}$ are of the form (3.3.2) (with appropriate sizes). Asking Maple for generalized eigenvalue of the pair $(C_{P,0}, C_{P,1})$ gives

$$\begin{bmatrix} 2.80 + 0.0\,i \\ 2.60 + 0.0\,i \\ 1.90 + 0.0\,i \\ 1.85 + 0.0\,i \\ 1.75 + 0.0\,i \\ 1.70 + 0.0\,i \\ 1.0 + 0.0\,i \end{bmatrix}$$

as roots of P and the corresponding pair to $Q(x)$ gives

$$\begin{bmatrix} 3.0 + 0.0\,i \\ 2.80 + 0.0\,i \\ 1.31 + 0.0\,i \\ 1.33 + 0.0\,i \\ 1.45 + 0.0\,i \\ 1.50 + 0.0\,i \end{bmatrix}$$

The second step is to cluster the set of roots. Before clustering we add multiplicities of the roots to the list of roots.

$$R_P = [[1, 1], [1.7, 1], [1.75, 1], [1.85, 1], [1.9, 1], [2.6, 1], [2.8, 1]],$$

$$R_Q = [[1.31, 1], [1.33, 1], [1.45, 1], [1.50, 1], [2.8, 1], [3, 1]].$$

By applying `ClusterRoots` with $\sigma = 0.5$ we get

$$R_P = [[1, 1], [1.825000000, 4], [2.700000000, 2]],$$

$$R_Q = [[1.442500000, 4], [2.900000000, 2]]$$

The clustered roots of P and Q will give us the following bipartite graph

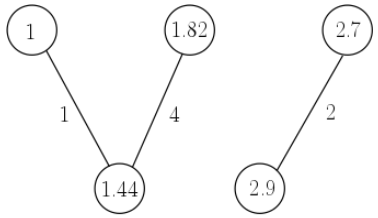

FIGURE 3.5: The bipartite graph corresponding to P and Q

.

Note that there are two MCM's given by

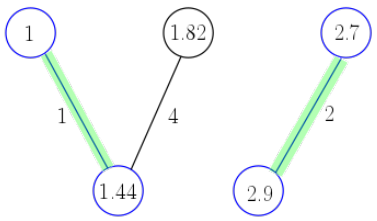

FIGURE 3.6: A Maximum Cardinality Matching for G, which is not a Maximum Weight Matching

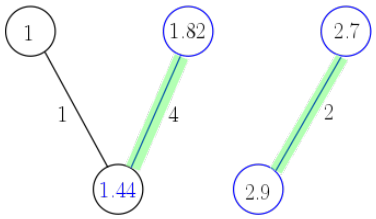

FIGURE 3.7: A Maximum Cardinality Matching for G, which is a Maximum Weight Matching as well.

Although Figure 3.6 is a MCM, it is not a maximum weight matching. This leads to a smaller degree in our approximate GCD. However, Figure 3.7 is a maximum weight matching which gives us a higher degree GCD.

Finally we need to find $G(x)$, $\tilde{P}$ and $\tilde{Q}$. Since the MWM given in Figure 3.7 contains two edges we have $\ell = 2$ and

$$G(x) = (x - \frac{1.82 + 1.44}{2})^4 \cdot (x - \frac{2.7 + 2.9}{2})^2 = (x - 1.63)^4 \cdot (x - 2.8)^2$$

which will be presented by

$$G_x = [1.63, 1.63, 1.63, 1.63, 2.8, 2.8, 0]$$

and

$$G_y = [0, 0, 0, 0, 0, 0, 38.98397700] .$$

In order to get $\tilde{P}$ and $\tilde{Q}$, we write:

$$\begin{aligned}
&r_1 = 1, r_{1+1} = 1.82, r_{1+2} = 2.7 \\
&d_1 = 1, d_{1+1} = 4, d_{1+2} = 2 \\
&s_{0+1} = 1.44, s_{0+2} = 2.9 \\
&d'_{0+1} = 4, d'_{0+2} = 2 \\
&w_1 = 4, w_2 = 2
\end{aligned}$$

This gives us

$$\tilde{P}(x) = G(x) \cdot (x-1)$$
$$\tilde{Q}(x) = G(x)$$

Also we can write

$$\tilde{P}_x = [1.63, 1.63, 1.63, 1.63, 2.8, 2.8, 1, 0]$$
$$\tilde{P}_y = [0, 0, 0, 0, 0, 0, 0, 38.98397700]$$

3.7.1 Another example of J. H. Wilkinson

In [41] we find the following example from filter design. This also appears as problem 8.41 in [16]. This is not the famous Wilkinson polynomial we discussed previously, or his less-famous second example polynomial as also discussed in [18], but rather a third example. The polynomial is given in the following non-standard form:

$$f(z) = \prod_{i=1}^{7} \left(z^2 + A_i z + B_i\right) + k \prod_{j=1}^{6} (z + C_j)^2 \ , \tag{3.7.1}$$

where the constants A_i, B_i, and C_j are given data; the constant k comes from other considerations. Wilkinson observes that for the data he uses, expansion into a monomial basis expression of this degree 14 polynomial is a bad idea, because the result is very ill-conditioned.

The constant k as given by Wilkinson is quite small, being $k = 1.38 \cdot 10^{-8}$. This suggests that using the zeros of the quadratic factors of the first term in the polynomial might give us good approximations of the zeros of $f(z)$ itself. This turns out to be a reasonable idea, but is not good enough for a perturbation series approach. Instead one must use numerics, such as are advocated in this paper. We modify the problem slightly, choosing $k = 1.38627474705020 \cdot 10^{-8}$ instead so that $f(z)$ has at least one multiple root (to fifteen figures). With the data given by Wilkinson, this gives

$$\begin{aligned} f(z) = &\left(z^2 + 2.008402247\, z + 1.008426206\right) \left(z^2 + 1.974225110\, z + 0.9749050168\right) \\ &\left(z^2 + 1.872661356\, z + 0.8791058345\right) \left(z^2 + 1.714140938\, z + 0.7375810928\right) \\ &\left(z^2 + 1.583160527\, z + 0.6279419845\right) \left(z^2 + 1.512571776\, z + 0.5722302977\right) \\ &\left(z^2 + 1.485030592\, z + 0.5513324340\right) - 1.38627474705020 \cdot 10^{-8}\, z^2 \\ &(z + 0.7015884551)^2\, (z + 0.6711668301)^2\, (z + 0.5892018711)^2\, (z + 1.084755941)^2 \\ &(z + 1.032359024)^2 \ . \end{aligned}$$

We compute all the zeros of each of the seven quadratic factors to fifteen figures, and compute the mean of these zeros as our 15th evaluation point (we need one more point than the degree in order to determine the polynomial). Then the value of $f(z)$ is computed at each of these 15 points (nodes).

We then compute the differentiation matrix for these nodes [3] and use that to evaluate $f'(z)$ at each of the nodes. This is one more value than is required to determine the degree 13 polynomial $f'(z)$ but the extra value does no harm.

We cluster the zeroes of $f(z)$, and find that there is only one nontrivial cluster, namely the conjugate pair $z = -0.7428866 \pm 2.0122 \times 10^{-13}\, i$, showing only those figures

that agree in the computed approximate conjugate pair. The mean of this cluster is $-0.742886654814304270 - 6.5 \times 10^{-18}\, i$ already showing the effect of taking the mean.

Computing the GCD with the values of $f'(z)$ on these nodes by the method of this paper (weighting the mean of the conjugate pair as twice as good as the corresponding zero of the derivative) gives $-0.742886654814185921 + 7.0 \times 10^{-17}\, i$ as the only common root.

3.8 Concluding Remarks

This paper solves the approximate GCD problem in Lagrange bases. Our algorithm follows the ideas in [19] and [37]. The results of this work are relevant to approximate GCD for a specific pseudometric, namely the root pseudometric.

A possible extension of this work may be approximate GCD in Lagrange bases with respect to a different metric (or pseudometric). Moreover, the main algorithm of this paper can be trivially extended to Hermite basis. The only difference will be the rootfinding step. In order to compute the roots one can use the companion pencil presented in [13].

One may wonder why clustering is used in $p(z)$ and separately in $q(z)$ to identify common roots, instead of simply considering the roots of $p(z)$ and $q(z)$ together and then clustering them (and thus identifying common zeros and hence the GCD). We feel that a significant feature of approximate multiple zeros arising from a *single* polynomial is *symmetry*: computing $p(z)(z-r)^m + \varepsilon$ results in a near-circular symmetric cluster near $z = r$. There is no reason to believe that the perturbation in $q(z)(z-r)^\ell + \delta$ would even be the same magnitude, and in any case symmetry would be lost. We admit that this is a heuristic, and does not account for *structured* perturbations $p(z)(z-r)^n + \varepsilon r(z)$; nonetheless we think this is a valuable heuristic and it has worked well for us in the examples that we have tried.

An interesting example, which we will describe in a future paper, is to cluster the roots of a Mandelbrot polynomial, defined as $p_0(z) = 0$ and $p_{n+1}(z) = zp_n^2(z) + 1$. These are studied by homotopy methods in [14] using the homotopy $f(z,t) = zp_n^2(z) + t$ which starts with double roots at $t = 0$: if one looks at the roots of $p_{n+1}(z)$ and clusters into double roots, do we get approximations of roots of $p_n(z)$? The answer is no, or not often, and then only in regions of sensitivity. But this example is, as mentioned in the previous paragraph, structured. We need to perform more experiments on this example before commenting further.

Finally, as it is described in this current paper, the clustering heuristic by distance, multiplicity, and symmetry does not take into any account that the points are actually supposed to be roots of a polynomial. Because it is based on the approximation

$$p(r + \Delta r) \approx \frac{p^{(m)}(r)}{m!} \Delta r^m \tag{3.8.1}$$

for an m-fold zero, we should in practice post-process the clustering results, estimate the mth derivative at a clustered root, and re-do the clustering with the "fuzz factors" of the heuristic replaced by the estimates resulting from the post-processing. This would increase our confidence that we had found an actual multiple root. Indeed one could imagine an iterative algorithm being based on this refinement step. We leave this to future work.

Acknowledgements

This work was supported by NSERC and the Ontario Research Centre for Computer Algebra. RMC thanks Laureano Gonzalez-Vega for a conversation in 1997 about the potential value of symmetry in clustering.

Bibliography

[1] A. Amiraslani. Dividing polynomials when you only know their values. In *Proceedings EACA*, pages 5–10, 2004.

[2] A. Amiraslani, R. M. Corless, L. Gonzalez-Vega, and A. Shakoori. Polynomial algebra by values. Technical report, Technical Report TR-04-01, Ontario Research Centre for Computer Algebra, 2004.

[3] A. Amiraslani, R. M. Corless, and M. Gunasingam. Differentiation matrices for univariate polynomials. *Numerical Algorithms*, pages 1–31, 2018.

[4] A. Amiraslani, R. M. Corless, and P. Lancaster. Linearization of matrix polynomials expressed in polynomial bases. *IMA J. Numer. Anal.*, 29(1):141–157, 2009.

[5] D. A. Aruliah, R. M. Corless, L. Gonzalez-Vega, and A. Shakoori. Geometric applications of the Bézout matrix in the Lagrange basis. In *Proceedings of the 2007 international workshop on Symbolic-numeric computation*, pages 55–64, 2007.

[6] R. Becker, M. Sagraloff, V. Sharma, J. Xu, and Ch. Yap. Complexity analysis of root clustering for a complex polynomial. In *Proceedings of the ACM on International Symposium on Symbolic and Algebraic Computation*, pages 71–78, 2016.

[7] B. Beckermann. The condition number of real Vandermonde, Krylov and positive definite Hankel matrices. *Numerische Mathematik*, 85(4):553–577, 2000.

[8] B. Beckermann and G. Labahn. When are two numerical polynomials relatively prime? *Journal of Symbolic Computation*, 26(6):677 – 689, 1998.

[9] B. Beckermann, G. Labahn, and A. C. Matos. On rational functions without Froissart doublets. *Numerische Mathematik*, 138(3):615–633, 2018.

[10] B. Botting, M. Giesbrecht, and J. May. Using Riemannian SVD for problems in approximate algebra. In *Proceedings of the International Workshop of Symbolic-Numeric Computation*, pages 209–219. Citeseer, 2005.

[11] J. C. Butcher, R. M. Corless, L. Gonzalez-Vega, and A. Shakoori. Polynomial algebra for Birkhoff interpolants. *Numerical Algorithms*, 56(3):319–347, 2011.

[12] J. M. Carnicer, Y. Khiar, and J. M. Peña. Optimal stability of the Lagrange formula and conditioning of the Newton formula. *Journal of Approximation Theory*, 2017.

[13] Chan, E. Y. S. and Corless, R. M. and Rafiee Sevyeri, L. Generalized Standard Triples for Algebraic Linearizations of Matrix Polynomials. *arXiv 1805.04488*, 2018.

[14] Eunice Y. S. Chan. A comparison of solution methods for Mandelbrot-like polynomials. *Master's thesis, University of Western Ontario*, 2016.

[15] R. M. Corless. Generalized companion matrices in the Lagrange basis. In *Proceedings EACA*, pages 317–322. Santander, Spain: Universidad de Cantabria, 2004.

[16] R. M. Corless and N. Fillion. *A Graduate Introduction to Numerical Methods: From the Viewpoint of Backward Error Analysis*. Springer, New York, 2013.

[17] R. M. Corless, P. M. Gianni, B. M. Trager, and S. M. Watt. The singular value decomposition for polynomial systems. In *Proceedings of the 1995 International Symposium on Symbolic and Algebraic Computation*, ISSAC '95, pages 195–207, New York, NY, USA, 1995. ACM.

[18] R. M. Corless and L. Rafiee Sevyeri. Stirling's original asymptotic series from a formula like one of Binet's and its evaluation by sequence acceleration. *Experimental Mathematics*, pages 1–8, 2019.

[19] R. M. Corless and L. Rafiee Sevyeri. Approximate GCD in a Bernstein basis. *Springer*, pages 77–91, 2020.

[20] R. M. Corless and L. Rafiee Sevyeri. The Runge Example for Interpolation and Wilkinson's Examples for Rootfinding. *SIAM Review*, 62(1) 231-243, 2018.

[21] R. M. Corless and S. M. Watt. Bernstein bases are optimal, but, sometimes, Lagrange bases are better. In *In Proceedings of SYNASC, Timisoara*, pages 141–153. MIRTON Press, 2004.

[22] R. M. Corless, A. Shakoori, D. A. Aruliah, and L. Gonzalez-Vega. Barycentric Hermite interpolants for event location in initial-value problems. *JNAIAM J. Numer. Anal. Indust. Appl. Math*, 3:1–16, 2008.

[23] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms*. MIT Press, Cambridge, MA; McGraw-Hill Book Co., Boston, MA, second edition, 2001.

[24] G. M. Diaz-Toca, M. Fioravanti, L. Gonzalez-Vega, and A. Shakoori. Using implicit equations of parametric curves and surfaces without computing them: Polynomial algebra by values. *Computer aided geometric design*, 30(1):116–139, 2013.

[25] G. M. Diaz-Toca and L. Gonzalez-Vega. Computing greatest common divisors and squarefree decompositions through matrix methods: The parametric and approximate cases. *Linear Algebra and its Applications*, 412(2):222 – 246, 2006.

[26] R. Duan and S. Pettie. Linear-time approximation for maximum weight matching. *Journal of the ACM (JACM)*, 61(1):1–23, 2014.

[27] R. T. Farouki and T. Goodman. On the optimal stability of the Bernstein basis. *Mathematics of Computation of the American Mathematical Society*, 65(216):1553–1566, 1996.

[28] S. Fortune. Polynomial root finding using iterated eigenvalue computation. In *Proceedings of the 2001 international symposium on Symbolic and algebraic computation*, pages 121–128, 2001.

[29] R. Imbach, V. Y. Pan, and Ch. Yap. Implementation of a near-optimal complex root clustering algorithm. In *International Congress on Mathematical Software*, pages 235–244. Springer, 2018.

[30] D. J. Jeffrey and R. M. Corless. *Linear algebra in Maple*, chapter 89. 2nd edition, 2013.

[31] E. Kaltofen, Z. Yang, and L. Zhi. Structured low rank approximation of a Sylvester matrix. In *Symbolic-numeric computation*, pages 69–83. Springer, 2007.

[32] N. K. Karmarkar and Y. N. Lakshman. Approximate polynomial Greatest Common Divisors and nearest singular polynomials. In *Proceedings of the 1996 International Symposium on Symbolic and Algebraic Computation*, ISSAC '96, pages 35–39, New York, NY, USA, 1996. ACM.

[33] N. K. Karmarkar and Y. N. Lakshman. On approximate gcds of univariate polynomials. *Journal of Symbolic Computation*, 26(6):653–666, 1998.

[34] J. L. Kelley. *General topology*. Courier Dover Publications, 2017.

[35] F. Le Gall. Powers of tensors and fast matrix multiplication. In *Proceedings of the 39th international symposium on symbolic and algebraic computation*, pages 296–303, 2014.

[36] Y. Nakatsukasa, O. SÃÍte, and L. Trefethen. The AAA algorithm for rational approximation. *SIAM Journal on Scientific Computing*, 40(3):A1494–A1522, 2018.

[37] V. Y. Pan. Numerical computation of a polynomial GCD and extensions. *Information and computation*, 167:71–85, 2001.

[38] N. Rezvani and R. M. Corless. Using weighted norms to find nearest polynomials satisfying linear constraints. In *Proceedings of the 2011 International Workshop on Symbolic-Numeric Computation*, SNC '11, pages 81–87, New York, NY, USA, 2011. ACM.

[39] A. Shakoori. The Bézout matrix in the Lagrange basis. In *Proceedings EACA*, pages 295–299. Citeseer, 2004.

[40] B. T. Smith. Error bounds for zeros of a polynomial based upon Gerschgorin's theorems. *Journal of the ACM (JACM)*, 17(4):661–674, 1970.

[41] J.H. Wilkinson. The evaluation of the zeros of ill-conditioned polynomials. part ii. *Numerische Mathematik*, 1(1):167–180, 1959.

[42] Z. Zeng. The approximate gcd of inexact polynomials. part I: a univariate algorithm. *Proceedings of the 2004 international symposium on Symbolic and algebraic computation, ACM*, pages 320–327, 2004.

[43] Z. Zeng. The numerical greatest common divisor of univariate polynomials. *Randomization, relaxation, and complexity in polynomial equation solving*, 556:187–217, 2011.

Chapter 4

Generalized Standard Triples for Algebraic Linearizations of Matrix Polynomials

4.1 Introduction

A *matrix polynomial* is usually defined as follows: "A matrix polynomial $\boldsymbol{P}(\lambda) \in F^{m\times n}[\lambda]$ is a polynomial in the variable λ with coefficients that are m by n matrices with entries from the field F." Typically an expression in the monomial basis $\phi_k(z) = z^k$ is given for $\boldsymbol{P}(\lambda)$, and often only *regular* matrix polynomials are considered, that is, with $m = n$ (we will use n for the dimension) and where $\det \boldsymbol{P}(\lambda)$ is not identically zero. Matrix polynomials have many applications and their study is of both classic and ongoing interest. See the classic work [17] and [20] for theory and applications.

In this paper, as is done in [2], we consider the case when *any* polynomial basis $\phi_k(\lambda)$ is used. We do require that the set $\{\phi_k(\lambda)\}$ for $0 \le k \le \ell$ forms a basis for polynomials of degree at most ℓ. Thus, we write our regular matrix polynomial as

$$\boldsymbol{P}(\lambda) = \sum_{k=0}^{\ell} \boldsymbol{P}_k \phi_k(\lambda)\,, \tag{4.1.1}$$

where the matrices $\boldsymbol{P}_k \in F^{n\times n}$ are square, and the *degree* of $\boldsymbol{P}(\lambda)$ is at most ℓ. The upper bound ℓ on the degree is also called the *grade*. The notion of "grade" is useful even for the monomial basis, but it is especially useful if the basis is an interpolational basis or the Bernstein basis $\phi_k(z) = B_k^n(z) = \binom{n}{k} z^k (1-z)^{n-k}$, when the degree of the polynomial may not be clear from the data.

For more information, consult [26]. See also [28], [20], and consult the seminal book [19]. Linearizations using different polynomial bases were first systematically studied in [2]. Some recent papers of interest include [5], [27], [11], [15], and [31]; this is a very active area. See also [1]. In that paper, standard triples for structured matrices are studied.

4.1.1 Organization of the Paper

In Section 4.1.2, we establish notation, give the definitions of algebraic linearization and of generalized standard triples. We define this last in Definition 4.1 with reference to the representation in Equation (4.1.10). In Section 4.1.3, we show how to use the generalized

standard triple in the construction of algebraic linearizations. We also give a proof, by construction of the necessary unimodular $\boldsymbol{E}$ and $\boldsymbol{F}$, that algebraic linearizations truly are linearizations.

In Section 4.2, we prove our main result, giving a universal expression for the generalized standard triples for the linearizations using the polynomial bases that appear in Sections 4.3.1, 4.3.2, and 4.3.3. In Section 4.3, we introduce all the polynomial bases by scalar examples; in that section, we also sneak in a new reversal of the Bernstein linearization and an outline of an apparently new proof that the Hermite interpolation basis linearization is, in fact, a linearization by explicitly constructing unimodular matrices $\boldsymbol{E}$ and $\boldsymbol{F}$ that bring the linearization to $\mathrm{diag}(\boldsymbol{P}(z), \boldsymbol{I}_n, \ldots, \boldsymbol{I}_n)$. In Section 4.4, we give the strict equivalence of generalized standard triples for the polynomial bases covered in this paper. We (redundantly) give specific proofs of our generalized standard triples formula, by using the Schur complement, in Section 4.5. We give matrix polynomial examples in Section 4.6. We sum up in the final section.

4.1.2 Notation and Definitions

Two matrix polynomials $\boldsymbol{P}_1(\lambda)$ and $\boldsymbol{P}_2(\lambda)$ are called *unimodularly equivalent* if there exist unimodular matrix polynomials (that is, matrices with constant nonzero determinant) $\boldsymbol{E}(\lambda)$ and $\boldsymbol{F}(\lambda)$ with $\boldsymbol{P}_1(\lambda) = \boldsymbol{E}(\lambda)\boldsymbol{P}_2(\lambda)\boldsymbol{F}(\lambda)$. A matrix pencil $\boldsymbol{L}(\lambda) := \lambda\boldsymbol{C}_1 - \boldsymbol{C}_0$ is called a *linearization* of the matrix polynomial $\boldsymbol{P}(\lambda)$ if both $\boldsymbol{C}_1$ and $\boldsymbol{C}_0$ are of dimension $N \geq n$ and $\boldsymbol{L}(\lambda)$ is unimodularly equivalent to the block diagonal matrix $\mathrm{diag}(\boldsymbol{P}(\lambda), \boldsymbol{I}_{N-n})$. Two linearizations $\boldsymbol{L}_m(\lambda)$ and $\boldsymbol{L}_\phi(\lambda)$ are called *strictly* equivalent if the corresponding matrices are equivalent in the following stronger sense: $\boldsymbol{C}_{1,m} = \boldsymbol{E}\boldsymbol{C}_{1,\phi}\boldsymbol{F}$ and $\boldsymbol{C}_{0,m} = \boldsymbol{E}\boldsymbol{C}_{0,\phi}\boldsymbol{F}$, with the same constant unimodular matrices $\boldsymbol{E}$ and $\boldsymbol{F}$.

Given a potential linearization $z\boldsymbol{C}_1 - \boldsymbol{C}_0$, it is possible to discover the matrices $\boldsymbol{E}$ and $\boldsymbol{F}$ by computing the *Hermite Form* of the linearization[1], with respect to the variable z, for instance by using a modestly sized example and a symbolic computation system such as Maple [32]. For instance, we can quickly find that if

$$\boldsymbol{E} = \begin{bmatrix} 1 & h_4 & h_3 & h_2 & h_1 \\ 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & -1 & -z \\ 0 & 0 & -1 & -z & -z^2 \\ 0 & -1 & -z & -z^2 & -z^3 \end{bmatrix} \tag{4.1.2}$$

where $h_5 = a_5$ and $h_k = a_k + zh_{k+1}$ for $k = 4, 3, 2, 1$ are the partial Horner evaluations of $p(z) = a_5z^5 + \cdots + a_0$, and if

$$\boldsymbol{F} = \begin{bmatrix} z^4 & 0 & 0 & 0 & 1 \\ z^3 & 0 & 0 & 1 & 0 \\ z^2 & 0 & 1 & 0 & 0 \\ z & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}, \tag{4.1.3}$$

[1]The Smith form, which is related, is also useful here; but we found that the Maple implementation of the Hermite Form gave a simpler answer, although we had to compute the matrix $\boldsymbol{F}$ separately ourselves because the Hermite form is upper triangular, not diagonal.

then for the second companion form

$$\boldsymbol{L}(z) = \begin{bmatrix} za_5 + a_4 & a_3 & a_2 & a_1 & a_0 \\ -1 & z & 0 & 0 & 0 \\ 0 & -1 & z & 0 & 0 \\ 0 & 0 & -1 & z & 0 \\ 0 & 0 & 0 & -1 & z \end{bmatrix}, \tag{4.1.4}$$

we have $\boldsymbol{E}\boldsymbol{L}(z)\boldsymbol{F} = \operatorname{diag}(p(z), 1, 1, 1, 1)$. Moreover, $\det \boldsymbol{E} = \pm 1$ and $\det \boldsymbol{F} = \pm 1$.

This quickly generalizes to matrix polynomials, and to arbitrary degree, establishing (as is well-known) that this form is a linearization.

There is a related idea to linearization, that of "companion pencil" $(\boldsymbol{A}, \boldsymbol{B})$, where the only requirement is that $\det \boldsymbol{P}(z) = \det(z\boldsymbol{B} - \boldsymbol{A})$. A companion pencil, therefore, has the same eigenvalues as the matrix polynomial. Companion pencils that are not linearizations do not necessarily preserve eigenvectors or elementary divisors, and are less useful than linearizations.

The usual *reversal*[2] of a matrix polynomial of degree at most ℓ is the polynomial $\operatorname{rev} \boldsymbol{P}(\lambda) = \lambda^\ell \boldsymbol{P}(\lambda^{-1})$. A linearization $\boldsymbol{L}(\lambda) = \lambda\boldsymbol{C}_1 - \boldsymbol{C}_0$ of $\boldsymbol{P}$ is called a *strong* linearization if $\operatorname{rev} \boldsymbol{L}(\lambda) = \boldsymbol{C}_1 - \lambda\boldsymbol{C}_0$ is also a linearization of $\operatorname{rev} \boldsymbol{P}(\lambda)$.

If $\boldsymbol{L}(z) = z\boldsymbol{C}_1 - \boldsymbol{C}_0 \in \mathbb{C}^{N\times N}[z]$—where usually $N = n\ell$ but not always; for Lagrange and Hermite interpolational bases some constructions use $N = (n+2)\ell$ and others $N = (n+1)\ell$—is a linearization of $\boldsymbol{P}(z)$, then as a necessary consequence $\det(\boldsymbol{P}(z)) = \det(\boldsymbol{L}(z)) = \det(z\boldsymbol{C}_1 - \boldsymbol{C}_0)$. The eigenvalues of $\boldsymbol{P}$ are thus computable from the generalized eigenvalues of $\boldsymbol{L}$. For instance, this can be done with `eig(C0,C1)` in Matlab, or `Eigenvalues(C[0],C[1])` in Maple if the matrix variables are defined appropriately (and are of complex floating-point type in Maple).

A *standard pair* $(\boldsymbol{X}, \boldsymbol{T})$ for a regular matrix polynomial $\boldsymbol{P}(\lambda)$ expressed in the monomial basis with coefficients $\boldsymbol{P}_k$ is defined in [19] or in [26] as having the following properties: $\boldsymbol{X}$ has dimension $n \times n\ell$, $\boldsymbol{T}$ has dimension $n\ell \times n\ell$,

$$\sum_{k=0}^{\ell} \boldsymbol{P}_k \boldsymbol{X} \boldsymbol{T}^k = 0\,, \tag{4.1.5}$$

and that the $n\ell$ by $n\ell$ matrix

$$\boldsymbol{Q} = \begin{bmatrix} \boldsymbol{X} \\ \boldsymbol{X}\boldsymbol{T} \\ \vdots \\ \boldsymbol{X}\boldsymbol{T}^{\ell-1} \end{bmatrix} \tag{4.1.6}$$

[2]This definition, which is standard, is particularly appropriate for the monomial basis. The coefficients of the reversed matrix polynomial in the monomial basis are simply the same matrices in reverse order. The notion of a reversal, however, is independent of the basis used, and indeed reversals can be done differently. In [9] for instance we find a slightly different definition of reversal, appropriate for computation in a Lagrange or Hermite interpolational basis, which maps an arbitrary finite point to infinity; this difference allows for greater numerical stability. We will introduce something similar for the Bernstein linearization in this present paper.

is nonsingular. We can then define a third matrix

$$\boldsymbol{Y} = \boldsymbol{Q}^{-1} \begin{bmatrix} \boldsymbol{0}_n \\ \vdots \\ \boldsymbol{0}_n \\ \boldsymbol{I}_n \end{bmatrix} \tag{4.1.7}$$

and say that the triple $(\boldsymbol{X}, \boldsymbol{T}, \boldsymbol{Y})$ is a *standard triple* for a *monic* $\boldsymbol{P}(\lambda)$. It is pointed out in [26] that monicity of $\boldsymbol{P}(\lambda)$ is not required for many of the formulæ to do with standard pairs (but is required for some).

Theorem 12.1.4 of [18] states that if there are matrices $\boldsymbol{X}$, $\boldsymbol{T}$, and $\boldsymbol{Y}$ of dimension $n \times n\ell$, $n\ell \times n\ell$, and $n\ell \times n$ for which

$$\boldsymbol{P}^{-1}(\lambda) = \boldsymbol{X}(\lambda \boldsymbol{I}_n - \boldsymbol{T})^{-1}\boldsymbol{Y} \tag{4.1.8}$$

then $(\boldsymbol{X}, \boldsymbol{T}, \boldsymbol{Y})$ is a standard triple for $\boldsymbol{P}(\lambda)$. This is also reported in Lemma 2 in [1]. There are two other representations of a matrix polynomial given a standard triple: the right canonical form, and the left canonical form. See Theorem 2.4 in [17]. However, we do not need those representations for algebraic linearization: it is the resolvent form above that we seek to generalize in this paper. The reason is that it is this formula that is used in the proof that algebraic linearizations can be performed when the component matrix polynomials are expressed in different bases.

A polynomial basis $\{\phi_k(z)\}_{k=0}^{\ell}$ for polynomials of degree at most ℓ (grade ℓ) is a set of polynomials each of degree at most ℓ for which there is a nonsingular matrix $\boldsymbol{\Phi}$ relating the polynomials $\phi_k(z)$ to the monomials $1, z, \ldots, z^{\ell}$. We can write this as

$$\begin{bmatrix} \phi_\ell(z) \\ \phi_{\ell-1}(z) \\ \vdots \\ \phi_0(z) \end{bmatrix} = \boldsymbol{\Phi} \begin{bmatrix} z^\ell \\ z^{\ell-1} \\ z^{\ell-2} \\ \vdots \\ 1 \end{bmatrix}. \tag{4.1.9}$$

Frequently, we want the $\ell \times \ell$ matrix that only goes up to degree $\ell - 1$. The matrix $\boldsymbol{\Phi}$ is called the change-of-basis matrix and is usually exponentially ill-conditioned in the dimension. For example, for the Bernstein polynomials $\phi_j^\ell(z) = \binom{\ell}{j} z^j (1-z)^{(\ell-j)}$ the change-of-basis matrix has entries $\phi_{i,j} = \binom{j}{i}/\binom{\ell}{j}$ and condition number $K_1 = K_\infty = (\ell+1)\binom{\ell}{s} 2^{\ell-s}$ where $s = \lceil (\ell-2)/3 \rceil$ (the notation $\lceil x \rceil$ means the ceiling of x, the least integer not smaller than x). A short computation shows $K \sim 3^{\ell+1}\sqrt{\ell/4\pi}$ as $\ell \to \infty$ and is thus exponentially growing with the dimension.

The constructions and definitions of standard triple discussed above are apparently tied to the monomial basis because of the powers $\boldsymbol{T}^k$ in Equation (4.1.6). We would like to relax this restriction and extend the notion of standard triple to other bases, and also to the non-monic case. In particular, we would like the following extension of Theorem 2.4 in [19] or Theorem 12.1.4 in [18] to be available: If a matrix $\boldsymbol{X} \in \mathbb{C}^{n \times N}$, the linearization $\boldsymbol{L}(z) = z\boldsymbol{C}_1 - \boldsymbol{C}_0 \in \mathbb{C}^{N \times N}[z]$, and a matrix $\boldsymbol{Y} \in \mathbb{C}^{N \times n}$ satisfies

$$\boldsymbol{P}^{-1}(z) = \boldsymbol{X}(z\boldsymbol{C}_1 - \boldsymbol{C}_0)^{-1}\boldsymbol{Y} \tag{4.1.10}$$

for $z \notin \Lambda(\boldsymbol{P})$ (the set of polynomial eigenvalues of $\boldsymbol{P}$), then $\boldsymbol{X}$, $\boldsymbol{L}(z)$, and $\boldsymbol{Y}$ form a *generalized standard triple* for $\boldsymbol{P}(\lambda)$. This obviously requires regularity of $\boldsymbol{P}$ because the formula contains $\boldsymbol{P}^{-1}(z)$.

Indeed, we simply require $\boldsymbol{L}(z)$ to be a linearization and take this extension as a *definition*. Such things exist, as we will demonstrate, and are useful, as shown in [7].

Definition 4.1. Matrices $\boldsymbol{X}$, $z\boldsymbol{C}_1 - \boldsymbol{C}_0$, and $\boldsymbol{Y}$ form a *generalized standard triple* for the regular matrix polynomial $\boldsymbol{P}(z)$ if $\boldsymbol{L}(z) = z\boldsymbol{C}_1 - \boldsymbol{C}_0$ is a linearization of $\boldsymbol{P}$ and Equation (4.1.10) holds.

Note that the matrices $\boldsymbol{X}$ and $\boldsymbol{Y}$ do not depend on z, but the linearization $\boldsymbol{L}(z)$ does, albeit only linearly; we could instead have chosen to use the words "standard quadruple" to mean $(\boldsymbol{X}, \boldsymbol{C}_1, \boldsymbol{C}_0, \boldsymbol{Y})$ where z does not appear of any of these matrices, but this quibble seems to be a matter of aesthetics only; we may use the term "triple" to refer to $\boldsymbol{X}$, the linearization, and $\boldsymbol{Y}$.

Remark 4.2. Note also that if $\boldsymbol{L}(z)$ is a linearization for $\boldsymbol{P}(z)$, then there exist unimodular matrices $\boldsymbol{E}$ and $\boldsymbol{F}$ with $\boldsymbol{F}^{-1}(z\boldsymbol{C}_1 - \boldsymbol{C}_0)^{-1}\boldsymbol{E}^{-1} = \operatorname{diag}(\boldsymbol{P}^{-1}(z), \boldsymbol{I}, \ldots, \boldsymbol{I})$. By premultiplying by $\boldsymbol{X}_p = [\boldsymbol{I}, 0, \ldots, 0]$ and postmultiplying by $\boldsymbol{Y}_p = [\boldsymbol{I}, 0, \ldots, 0]^T$, we may find $\boldsymbol{X} = \boldsymbol{X}_p\boldsymbol{F}^{-1}$ and $\boldsymbol{Y} = \boldsymbol{E}^{-1}\boldsymbol{Y}_p$ so that Equation (4.1.10) holds. Thus, as a referee pointed out, the generalized standard triples may be read off from the proof that $\boldsymbol{L}(z)$ is indeed a linearization (with a little work).

It is not clear that if Equation (4.1.10) holds then $\boldsymbol{L}(z)$ is necessarily a linearization of $\boldsymbol{P}(z)$. We do have, however, the following:

Lemma 4.3. *If a generalized standard triple exists as in Equation* (4.1.10)*, then the matrix pencil* $z\boldsymbol{C}_1 - \boldsymbol{C}_0$ *is at least a companion pencil for* $\boldsymbol{P}(z)$*.*

Proof. The norm of the resolvent $\|\boldsymbol{P}^{-1}(z)\|$ will be large if and only if $\|\,(z\boldsymbol{C}_1 - \boldsymbol{C}_0)^{-1}\,\|$ is large. □

For various reasons, we usually do not wish to invert a "leading coefficient" here; for instance, if the polynomial basis is not degree-graded, e.g. for the Bernstein basis, then in order to even look at the true leading coefficient, we have to form a particular linear combination of the existing coefficients. In floating-point arithmetic, rounding errors can disguise the rank of the resulting matrix, hence our interest in the generalization.

All we will need for the purposes of this paper is that the representation displayed in Equation (4.1.10) holds. The representation itself is what is useful in the recursive construction of algebraic linearizations.

If $\boldsymbol{X}$, $z\boldsymbol{C}_1 - \boldsymbol{C}_0$, and $\boldsymbol{Y}$ form a generalized standard triple according to our definition, then so do $\boldsymbol{X}\boldsymbol{U}$, $\boldsymbol{U}^{-1}(z\boldsymbol{C}_1 - \boldsymbol{C}_0)\boldsymbol{V}$, and $\boldsymbol{V}^{-1}\boldsymbol{Y}$ for any nonsingular matrices $\boldsymbol{U}$ and $\boldsymbol{V}$ of dimension N by N.

Several similarities are used very frequently. For convenience, we describe two of the most common explicitly here.

Lemma 4.4 (Flipping)**.** *Put* $\boldsymbol{J}$ *as the* $N \times N$ *"anti-identity", also called the sip matrix, for standard involutory permutation,* $\boldsymbol{J}_{i,j} = 0$ *unless* $i + j = N + 1$ *when* $\boldsymbol{J}_{i,N+1-i} = 1$*. Then* $\boldsymbol{J}^2 = \boldsymbol{I}$ *and the "flipped" linearization* $\boldsymbol{L}_{\boldsymbol{F}}(z) = \boldsymbol{J}(z\boldsymbol{C}_1 - \boldsymbol{C}_0)\boldsymbol{J}$ *has in its generalized standard triple the matrices* $\boldsymbol{X}_{\boldsymbol{F}} = \boldsymbol{X}\boldsymbol{J}$ *and* $\boldsymbol{Y}_{\boldsymbol{F}} = \boldsymbol{J}\boldsymbol{Y}$*.*

Proof. Immediate. □

Remark 4.5. Flipping switches both the order of the equations and the order of the variables. It obviously does not change eigenvalues. Flipping, transposition, and flipping-with-transposition give four common equivalent linearizations [34].

4.1.3 Algebraic Linearizations

An *algebraic linearization* $(\boldsymbol{H}, \boldsymbol{D}_H)$, as referred to in the title of this present note, is defined in [7] as a linearization $(\boldsymbol{H}, \boldsymbol{D}_H)$ of a matrix polynomial $\boldsymbol{h}(\lambda) = \lambda \boldsymbol{a}(\lambda)\boldsymbol{d}_0\boldsymbol{b}(\lambda) + \boldsymbol{c}$ constructed recursively from linearizations $(\boldsymbol{A}, \boldsymbol{D}_A)$ and $(\boldsymbol{B}, \boldsymbol{D}_B)$ of the lower-degree component matrix polynomials $\boldsymbol{a}(\lambda)$ and $\boldsymbol{b}(\lambda)$, together with constant matrices $\boldsymbol{d}_0$ and $\boldsymbol{c}$. The paper [7] did not give an explicit unimodular pair $(\boldsymbol{E}_H, \boldsymbol{F}_H)$ that reduced the linearization to $\operatorname{diag}(z\boldsymbol{a}(z)\boldsymbol{d}_0\boldsymbol{b}(z) + \boldsymbol{c}_0, \boldsymbol{I}_n, \ldots, \boldsymbol{I}_n)$, proving that the construction actually gave a linearization, so we give a method to construct them here. Without loss of generality we take $\boldsymbol{d}_0 = \boldsymbol{I}_n$.

Theorem 4.6. *If the n by n matrix polynomial $\boldsymbol{a}(z)$ has linearization $(\boldsymbol{A}, \boldsymbol{D}_A)$ with unimodular pair $(\boldsymbol{E}_A, \boldsymbol{F}_A)$ and if the n by n matrix polynomial $\boldsymbol{b}(z)$ has linearization $(\boldsymbol{B}, \boldsymbol{D}_B)$ with unimodular pair $(\boldsymbol{E}_B, \boldsymbol{F}_B)$ then the pencil $z\boldsymbol{D}_H - \boldsymbol{H}$ is a linearization of $\boldsymbol{h}(z) = z\boldsymbol{a}(z)\boldsymbol{b}(z) + \boldsymbol{C}$, where the matrices $\boldsymbol{D}_H$ and $\boldsymbol{H}$ are given as follows:*

$$\boldsymbol{D}_H = \begin{bmatrix} \boldsymbol{D}_A & & \\ & \boldsymbol{I}_n & \\ & & \boldsymbol{D}_B \end{bmatrix} \tag{4.1.11}$$

and

$$\boldsymbol{H} = \begin{bmatrix} \boldsymbol{A} & \boldsymbol{0}_{N_A,n} & -\boldsymbol{Y}_A\boldsymbol{c}\boldsymbol{X}_B \\ -\boldsymbol{X}_A & \boldsymbol{0}_n & \boldsymbol{0}_{n,N_B} \\ \boldsymbol{0}_{N_B,N_A} & -\boldsymbol{Y}_B & \boldsymbol{B} \end{bmatrix}. \tag{4.1.12}$$

Here $\boldsymbol{X}_A = [\boldsymbol{I}_n, 0, \ldots, 0]\boldsymbol{F}_A^{-1}$ $\boldsymbol{Y}_A = \boldsymbol{E}_A^{-1}[\boldsymbol{I}_n, 0, \ldots, 0]^T$ and likewise $\boldsymbol{X}_B = [\boldsymbol{I}_n, 0, \ldots, 0]\boldsymbol{F}_B^{-1}$ and $\boldsymbol{Y}_B = \boldsymbol{E}_A^{-1}[\boldsymbol{I}_n, 0, \ldots, 0]^T$ give the elements of the (generalized) standard triples for $\boldsymbol{a}(z)$ and $\boldsymbol{b}(z)$.

Proof. We first construct

$$\boldsymbol{E}_1 = \begin{bmatrix} \boldsymbol{E}_A & & \\ & \boldsymbol{I}_n & \\ & & \boldsymbol{E}_B \end{bmatrix} \tag{4.1.13}$$

and

$$\boldsymbol{F}_1 = \begin{bmatrix} \boldsymbol{F}_A & & \\ & \boldsymbol{I}_n & \\ & & \boldsymbol{F}_B \end{bmatrix}. \tag{4.1.14}$$

Applying them we get

$$\boldsymbol{E}_1(z\boldsymbol{D}_H - \boldsymbol{H})\boldsymbol{F}_1 = \begin{bmatrix} \boldsymbol{a}(z) & & & \boldsymbol{E}_A\boldsymbol{Y}_A\boldsymbol{c}\boldsymbol{X}_B\boldsymbol{F}_B & \\ & \boldsymbol{I}_{N_A-n} & & & \\ \boldsymbol{X}_A\boldsymbol{F}_A & & z\boldsymbol{I}_n & & \\ & & \boldsymbol{E}_B\boldsymbol{Y}_B & \boldsymbol{b}(z) & \\ & & & & \boldsymbol{I}_{N_B-n} \end{bmatrix}. \tag{4.1.15}$$

Simplifying and using the definitions of the matrices appearing in the standard triples, we get

$$\begin{bmatrix} \boldsymbol{a}(z) & & \boldsymbol{c} & \\ & \boldsymbol{I}_{N_A-n} & & \\ \boldsymbol{I}_n & & z\boldsymbol{I}_n & \\ & & \boldsymbol{I}_n & \boldsymbol{b}(z) \\ & & & \boldsymbol{I}_{N_B-n} \end{bmatrix}. \tag{4.1.16}$$

Multiplying on the right by $\operatorname{diag}(\boldsymbol{I}_{N_A}, \boldsymbol{b}(z)\cdot\boldsymbol{b}^{-1}(z), \boldsymbol{I}_{N_B})$ (which we can do except when z is an eigenvalue of $\boldsymbol{b}$) we get

$$\begin{bmatrix} \boldsymbol{a}(z) & & & \boldsymbol{c} & \\ & \boldsymbol{I}_{N_A-n} & & & \\ \boldsymbol{I}_n & & z\boldsymbol{b}(z) & & \\ & & \boldsymbol{b}(z) & \boldsymbol{b}(z) & \\ & & & & \boldsymbol{I}_{N_B-n} \end{bmatrix}. \tag{4.1.17}$$

An elementary block column operation (recorded as the appropriate block elementary matrix multiplication) gets us

$$\begin{bmatrix} \boldsymbol{a}(z) & & -\boldsymbol{c} & \boldsymbol{c} & \\ & \boldsymbol{I}_{N_A-n} & & & \\ \boldsymbol{I}_n & & z\boldsymbol{b}(z) & & \\ & & 0 & \boldsymbol{b}(z) & \\ & & & & \boldsymbol{I}_{N_B-n} \end{bmatrix}. \tag{4.1.18}$$

Multiplying the block containing $z\boldsymbol{b}(z)$ by $\boldsymbol{a}(z)$ and subtracting it from the first block row (recording this operation in a factor on the left) we get

$$\begin{bmatrix} 0 & & -z\boldsymbol{a}(z)\boldsymbol{b}(z)-\boldsymbol{c} & \boldsymbol{c} & \\ & \boldsymbol{I}_{N_A-n} & & & \\ \boldsymbol{I}_n & & z\boldsymbol{b}(z) & & \\ & & 0 & \boldsymbol{b}(z) & \\ & & & & \boldsymbol{I}_{N_B-n} \end{bmatrix}. \tag{4.1.19}$$

Interchanging columns and recording it on the right, and finally pulling out a factor $\boldsymbol{b}(z)$ by a column multiplication, we arrive at

$$\begin{bmatrix} -z\boldsymbol{a}(z)\boldsymbol{b}(z)-\boldsymbol{c} & & & \boldsymbol{c}\boldsymbol{b}^{-1}(z) & \\ & \boldsymbol{I}_{N_A-n} & & & \\ z\boldsymbol{b}(z) & & \boldsymbol{I}_n & & \\ & & 0 & \boldsymbol{I}_n & \\ & & & & \boldsymbol{I}_{N_B-n} \end{bmatrix}. \tag{4.1.20}$$

The blocks $z\boldsymbol{b}(z)$ and $\boldsymbol{c}\boldsymbol{b}^{-1}$ can be removed using a block column operation and a block row operation, and we have arrived at the desired form (apart from a sign, which can be absorbed either into the first row of $\boldsymbol{E}_H$ or the first column of $\boldsymbol{F}_H$). All that remains is to check that the constructed factors (recording all our operations) $\boldsymbol{E}_H$ and $\boldsymbol{F}_H$ are unimodular. This is so because we put both a factor $\boldsymbol{b}(z)$ and a factor $\boldsymbol{b}^{-1}(z)$ into the matrices on the right and otherwise nothing depending on z, and on the left only constant-determinant factors. This completes the proof. □

Remark 4.7. The generalized standard triple for the algebraic linearization has $\boldsymbol{X}_H = [0, 0, \boldsymbol{X}_B]$ and $\boldsymbol{Y}_H = [\boldsymbol{Y}_A^T, 0, 0]^T$. For these linearizations, there is no notion of expressing 1 as a linear combination of anything, because this formulation is free of polynomial bases.

Algebraic linearizations offer a new, potentially more numerically stable, class of linearizations. The recursive construction of algebraic linearizations relies on the generalized standard triples of each of the component matrix polynomials, and (as does the unrelated paper [31]) allows different polynomial bases to be used for each component. This present note provides some explicit formulas for generalized standard triples in

various bases, for reference. As one reviewer points out, these formulas *could* simply be obtained by reading the proofs that these linearizations are indeed linearizations; one purpose of this paper is simply convenience.

4.2 Expressing 1 in the basis gives the triple

If the $\phi_k(x)$, $0 \le k \le \ell - 1$ form a basis, we may express the polynomial 1 in that basis: then $1 = \sum_{k=0}^{\ell-1} e_k \phi_k(z)$ defines the coefficients e_k uniquely. Putting

$$\boldsymbol{X} = \begin{bmatrix} e_{\ell-1} & e_{\ell-2} & \cdots & e_1 & e_0 \end{bmatrix} \otimes \boldsymbol{I} \tag{4.2.1}$$

for an appropriate choice of basis always gives our generalized standard triple $\boldsymbol{P}^{-1}(z) = \boldsymbol{X}(z\boldsymbol{C}_1 - \boldsymbol{C}_0)^{-1}\boldsymbol{Y}$ with

$$\boldsymbol{Y} = \begin{bmatrix} \boldsymbol{I}_n & \boldsymbol{0}_n & \boldsymbol{0}_n & \cdots & \boldsymbol{0}_n \end{bmatrix}^T . \tag{4.2.2}$$

We prove this below for all the elementary linearizations we use in this paper.

Theorem 4.8. *The generalized standard triple for $\boldsymbol{P}(z)$ in the basis $\boldsymbol{\Phi}$ is given as described above.*

Proof. The following proof, which uses an idea of an anonymous referee, is simpler than our original one. For each of the polynomial bases we examine in this paper, the linearization satisfies either

$$\boldsymbol{L}(z) \begin{bmatrix} \phi_{\ell-1}(z)\boldsymbol{I}_n \\ \phi_{\ell-2}(z)\boldsymbol{I}_n \\ \vdots \\ \phi_0(z)\boldsymbol{I}_n \end{bmatrix} = \begin{bmatrix} \boldsymbol{I}_n \\ \boldsymbol{0}_n \\ \vdots \\ \boldsymbol{0}_n \end{bmatrix} \boldsymbol{P}(z) , \tag{4.2.3}$$

for degree-graded bases, or similar statements for Bernstein bases and Lagrange and Hermite interpolational bases, as follows. For the Bernstein basis, the polynomial elements in the vector on the left are multiples of $B_j^{\ell-1}(z)$: $[\ell/1 \cdot B_{\ell-1}^{\ell-1}(z), \ell/2 \cdot B_{\ell-1}^{\ell-1}(z), \ldots, \ell/\ell \cdot B_0^{\ell-1}(z)]^T$. For the Lagrange basis, the vector on the left is $[w(z), \ell_0(z), \ell_1(z), \ldots, \ell_\ell(z)]^T$. For the Hermite interpolational basis, it is the same as for the Lagrange but with the Lagrange basis elements replaced with the Hermite interpolational basis elements.

Premultiplying by $\boldsymbol{L}^{-1}(z)$ and post-multiplying by $\boldsymbol{P}^{-1}(z)$, we have

$$\boldsymbol{L}^{-1}(z) \begin{bmatrix} \boldsymbol{I}_n \\ \boldsymbol{0}_n \\ \vdots \\ \boldsymbol{0}_n \end{bmatrix} = \begin{bmatrix} \phi_{\ell-1}(z)\boldsymbol{I}_n \\ \phi_{\ell-2}(z)\boldsymbol{I}_n \\ \vdots \\ \phi_0(z)\boldsymbol{I}_n \end{bmatrix} \boldsymbol{P}^{-1}(z) . \tag{4.2.4}$$

If $1 = \sum_{k=0}^{\ell-1} e_k \phi_k(z)$ is the expression of 1 in that basis, then premultiplying both sides by

$$\boldsymbol{X} = \begin{bmatrix} e_{\ell-1}\boldsymbol{I}_n & e_{\ell-2}\boldsymbol{I}_n & \ldots & e_0\boldsymbol{I}_n \end{bmatrix}$$

gives the theorem. Compare also Remark 4.2 which gives another formula for $\boldsymbol{X}$ and $\boldsymbol{Y}$.

Note that in the Bernstein, Lagrange, and Hermite interpolational cases, 1 can be expressed as a linear combination of the elements given; for Lagrange and Hermite the coefficient of $w(z)$ is 0. These differences between this and the non degree-graded bases will be brought out in the examples, and the individual proofs. See in particular

Equation (4.4.9) and Equation (4.4.10) for Bernstein and Lagrange forms, respectively. The Hermite interpolational case is very much like the Lagrange case. In all cases, expressing 1 as a linear combination of elements proves to be crucial. □

Remark 4.9. There are linearizations not explicitly considered in this paper; for instance, a referee has pointed out that when a matrix polynomial is expressed in a basis where the elements satisfy a linear recurrence, then there is an automatic way to build what is called a CORK linearization. See [20] and [35] for details.

In what follows we examine specific cases in detail and supply specific proofs for each basis. Indeed, much of the utility of this paper is simply writing down those details, which will allow easier programming for the uses of these generalized standard triples.

4.3 Scalar examples of generalized standard triples

In this section, we tabulate generalized standard triples for four classes of linearizations. We do so by scalar examples, leaving the matrix polynomial case to Section 4.6. In contrast, in Section 4.5 where we gave proofs, we do so in full generality.

In the special case $n = 1$ and when the monomial basis is used, a linearization is usually simplified by dividing by the leading coefficient, making the result monic and the second matrix of the pair just becomes the identity. The remaining matrix is called a "companion matrix" or Frobenius companion[3]. Thus finding roots of a scalar polynomial can be done by finding eigenvalues of the companion matrix. Kublanovskaya calls these "accompanying pencils" in [24]. For bases other than the monomial, the unfortunate nomenclature "colleague matrix" or "comrade matrix" is also used. This nomenclature hinders citation search and we prefer "generalized companion", if a distinction is needed. See [28].

Construction of a linearization from a companion matrix is, when possible at all, a simple matter of the Kronecker (tensor) product: given $\boldsymbol{C}_1$, $\boldsymbol{C}_0 \in \mathbb{C}^{n\times n}$, take $\widetilde{\boldsymbol{C}_1} = \boldsymbol{C}_1 \otimes \boldsymbol{I}_n$ and then replace each block $p_k\boldsymbol{I}_n$ with the corresponding matrix coefficient $\boldsymbol{P}_k \in \mathbb{C}^{r\times r}$ (the first p_k, in $p_k\boldsymbol{I}_n$, is the symbolic coefficient from $p(z) = \sum_{k=0}^{\ell} p_k\phi_k(z)$; the matrix coefficient $\boldsymbol{P}_k \in \mathbb{C}^{r\times r}$ is from $\boldsymbol{P}(z) = \sum_{k=0}^{\ell} \boldsymbol{P}_k\phi_k(z)$.) This will be clearer by example.

4.3.1 Bases with three-term recurrence relations

The monomial basis, the shifted monomial basis, the Taylor basis, the Newton interpolational bases, and many common orthogonal polynomial bases all have three-term recurrence relations that, except for initial cases, can be written

$$z\phi_k(z) = \alpha_k\phi_{k+1}(z) + \beta_k\phi_k(z) + \gamma_k\phi_{k-1}(z)\,. \tag{4.3.1}$$

In all cases, we have $\alpha_k \neq 0$. For instance, the Chebyshev polynomial recurrence is usually written $T_{n+1}(z) = 2zT_n(z) - T_{n-1}(z)$ but is easily rewritten in the above form by isolating $zT_n(z)$, and all Chebyshev $\alpha_k = 1/2$ for $k > 1$. We give a selection in Table 4.1, and refer the reader to section 18.9 of the Digital Library of Mathematical Functions (`dlmf.nist.gov`) for more. See also [16].

[3]The *Frobenius form* of a matrix is related, but different: see for instance [33].

$\phi_k(z)$	Name	α_k	β_k	γ_k	ϕ_0	ϕ_1
z^k	monomial	1	0	0	1	z
$(z-a)^k$	shifted monomial	1	a	0	1	$z-a$
$(z-a)^k/k!$	Taylor	$n+1$	a	0	1	$z-a$
$\prod_{j=0}^{k-1}(z-\tau_j)$	Newton interpolational	1	τ_n	0	1	$z-\tau_0$
$T_k(z)=\cos\left(k\cos^{-1}(z)\right)$	Chebyshev	$1/2$	0	$1/2$	1	z
$P_k(z)$	Legendre	$(k+1)/(2k+1)$	0	$k/(2k+1)$	1	z

TABLE 4.1: A short list of three-term recurrence relations for some important polynomial bases discussed in Section 4.3.1. For a more comprehensive list, see The Digital Library of Mathematical Functions. These relations and others are coded in Walter Gautschi's packages OPQ and SOPQ [16] and in the `MatrixPolynomialObject` implementation package in Maple (see [21]).

For all such bases, we have the linearization[4]

$$\boldsymbol{C}_1 = \left[\begin{array}{c|ccc} \frac{p_5}{\alpha_4} & & & \\ \hline 1 & & & \\ & 1 & & \\ & & 1 & \\ & & & 1 \end{array}\right], \tag{4.3.2}$$

$$\boldsymbol{C}_0 = \left[\begin{array}{c|cccc} -p_4+\frac{\beta_4}{\alpha_4}p_5 & -p_3+\frac{\gamma_4}{\alpha_4}p_5 & -p_2 & -p_1 & -p_0 \\ \hline \alpha_3 & \beta_3 & \gamma_3 & & \\ & \alpha_2 & \beta_2 & \gamma_2 & \\ & & \alpha_1 & \beta_1 & \gamma_1 \\ & & & \alpha_0 & \beta_0 \end{array}\right], \tag{4.3.3}$$

(remember that $\alpha_k \neq 0$) and

$$\boldsymbol{X} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \tag{4.3.4}$$

$$\boldsymbol{Y} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \end{bmatrix}^{\mathrm{T}}. \tag{4.3.5}$$

For instance, a *flipped and transposed* linearization of this class for the Chebyshev case is[5]

$$\boldsymbol{L}(z) = \begin{bmatrix} z & -\frac{1}{2} & & & p_0 \\ -1 & z & -\frac{1}{2} & & p_1 \\ & -\frac{1}{2} & z & -\frac{1}{2} & p_2 \\ & & -\frac{1}{2} & z & p_3+p_5 \\ & & & -\frac{1}{2} & 2zp_5+p_4 \end{bmatrix} \tag{4.3.6}$$

[4] For exposition, we follow Peter Lancaster's dictum, namely that the 5×5 case almost always gives the idea.

[5] For the matrix polynomial case, each $\boldsymbol{P}_k$ would be transposed. If we had started with $\boldsymbol{P}^T(z)$ then by flipping and transposing we get back to a linearization for $\boldsymbol{P}$.

has flipped and transposed $\boldsymbol{X} = \begin{bmatrix}0 & 0 & 0 & 0 & 1\end{bmatrix}$, $\boldsymbol{Y} = \begin{bmatrix}1 & 0 & 0 & 0 & 0\end{bmatrix}^{\mathrm{T}}$. As another instance, a Newton interpolational basis on the nodes τ_0, τ_1, …, τ_5 has a linearization

$$z\begin{bmatrix} p_5 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & & 1 & \\ & & & & 1 \end{bmatrix} - \begin{bmatrix} -p_4 + \tau_4 p_5 & -p_3 & -p_2 & -p_1 & -p_0 \\ 1 & \tau_3 & & & \\ & 1 & \tau_2 & & \\ & & 1 & \tau_1 & \\ & & & 1 & \tau_0 \end{bmatrix}. \tag{4.3.7}$$

The corresponding linearization for matrix polynomials of this grade is

$$z\begin{bmatrix} \boldsymbol{P}_5 & & & & \\ & \boldsymbol{I}_n & & & \\ & & \boldsymbol{I}_n & & \\ & & & \boldsymbol{I}_n & \\ & & & & \boldsymbol{I}_n \end{bmatrix} - \begin{bmatrix} -\boldsymbol{P}_4 + \tau_4 \boldsymbol{P}_5 & -\boldsymbol{P}_3 & -\boldsymbol{P}_2 & -\boldsymbol{P}_1 & -\boldsymbol{P}_0 \\ \boldsymbol{I}_n & \tau_3 \boldsymbol{I}_n & & & \\ & \boldsymbol{I}_n & \tau_2 \boldsymbol{I}_n & & \\ & & \boldsymbol{I}_n & \tau_1 \boldsymbol{I}_n & \\ & & & \boldsymbol{I}_n & \tau_0 \boldsymbol{I}_n \end{bmatrix}. \tag{4.3.8}$$

4.3.2 The Bernstein basis

The set of polynomials $\{B_k^\ell(z)\}_{k=0}^\ell = \binom{\ell}{k} z^k (1-z)^{\ell-k}$ is a set of $\ell+1$ polynomials each of exact degree ℓ that together forms a basis for polynomials of degree at most ℓ (grade ℓ). Bernstein polynomimals have many applications, for example in Computer Aided Geometric Design (CAGD), and many important properties including that of optimal condition number over all bases positive on $[0, 1]$. They do not satisfy a simple three term recurrence relation of the form discussed in Section 4.3.1, although they satisfy an interesting and useful "degree-elevation" recurrence, namely

$$(j+1)B_{j+1}^n(z) + (n-j)B_j^n(z) = nB_j^{n-1}(z)\,, \tag{4.3.9}$$

which specifically demonstrates that a sum of Bernstein polynomials of degree n might actually have degree strictly less than n. See [12], [13], and [14] for more details of Bernstein bases.

A Bernstein linearization for $p_5(z) = \sum_{k=0}^5 p_k B_k^5(z)$ is

$$\boldsymbol{C}_1 = \begin{bmatrix} -p_4 + \frac{1}{5}p_5 & -p_3 & -p_2 & -p_1 & -p_0 \\ 1 & \frac{2}{4} & & & \\ & 1 & \frac{3}{3} & & \\ & & 1 & \frac{4}{2} & \\ & & & 1 & \frac{5}{1} \end{bmatrix}, \tag{4.3.10}$$

$$\boldsymbol{C}_0 = \begin{bmatrix} -p_4 & -p_3 & -p_2 & -p_1 & -p_0 \\ 1 & 0 & & & \\ & 1 & 0 & & \\ & & 1 & 0 & \\ & & & 1 & 0 \end{bmatrix}, \tag{4.3.11}$$

$$\boldsymbol{X} = \begin{bmatrix} \frac{1}{5} & \frac{2}{5} & \frac{3}{5} & \frac{4}{5} & \frac{5}{5} \end{bmatrix}, \tag{4.3.12}$$

$$\boldsymbol{Y} = \begin{bmatrix}1 & 0 & 0 & 0 & 0\end{bmatrix}^{\mathrm{T}}. \tag{4.3.13}$$

For a construction of the $\boldsymbol{E}$ and $\boldsymbol{F}$ that show this is a linearization, see [2]. This is a *strong* linearization, as we will explicitly see shortly. We have $p^{-1}(z) = \boldsymbol{X}(z\boldsymbol{C}_1 - \boldsymbol{C}_0)^{-1}\boldsymbol{Y}$ if $p(z) \neq 0$. This linearization was first analyzed in [22] and [23] and has been further studied and generalized in [29]. One of the present authors independently invented and implemented a version of this linearization in Maple (except using $\boldsymbol{P}^{\mathrm{T}}(z)$, and flipped from the above form) in about 2004. For a review of Bernstein linearizations, see the aforementioned [29]. For a proof of their numerical stability, see the original thesis [22]. The standard triple is, we believe, new to this paper.

4.3.2.1 A new reversal

As a further novelty, we here advocate a slightly different *reversal*, namely $\operatorname{rev} p(z) = (z+1)^n p(1/(z+1))$ of a polynomial of degree at most n expressed in a Bernstein basis, instead of the standard reversal $z^n p(1/z)$. This new reversal has a slight numerical advantage if all the coefficients of $p(z)$ are the same sign. We also give a proof that the linearization of this reversal is the corresponding reversal of the linearization, thus giving a new independent proof that the linearization is a strong one.

A short computation shows that if

$$p(z) = \sum_{k=0}^{n} c_k B_k^n(z) \tag{4.3.14}$$

then

$$\operatorname{rev} p(z) = (z+1)^n p\left(\frac{1}{z+1}\right) = \sum_{k=0}^{n} d_k B_k^n(z) \tag{4.3.15}$$

where

$$d_k = \sum_{j=0}^{k} \binom{k}{j} c_{n-j}\,, \tag{4.3.16}$$

whereas the coefficients of the standard reversal are, in contrast,

$$e_k = \sum_{m=0}^{n-k} (-1)^m \binom{n-k}{m} c_{n-m-k} \tag{4.3.17}$$

which has introduced sign changes, which may fail to preserve numerical stability if all the c_k are of one sign. A further observation is that the coefficient d_0 only involves c_n, while e_0 involves all c_k; d_1 involves c_n and c_{n-1} while e_1 involves all but c_0, and so on; in that sense, this new reversal has a more analogous behaviour to the monomial basis reversal, which simply reverses the list of coefficients.

For interest, we note that if $(\boldsymbol{A}, \boldsymbol{B})$ is a linearization for $p(z)$ so that $p(z) = \det(z\boldsymbol{B} - \boldsymbol{A})$, then reversing the linearization by this transformation is not a matter of simply interchanging $\boldsymbol{B}$ and $\boldsymbol{A}$:

$$\begin{aligned}(z+1)^n p\left(\frac{1}{z+1}\right) &= (z+1)^n \det\left(\frac{1}{z+1}\boldsymbol{B} - \boldsymbol{A}\right) \\ &= \det(\boldsymbol{B} - (z+1)\boldsymbol{A}) \\ &= \det(\boldsymbol{B} - \boldsymbol{A} - z\boldsymbol{A})\end{aligned} \tag{4.3.18}$$

and so the corresponding reversed linearization is $(\boldsymbol{A}, \boldsymbol{B} - \boldsymbol{A})$. The sign change is of no importance.

Suppose that the Bernstein linearization of $p(z)$ is $(\boldsymbol{A}, \boldsymbol{B})$ and that the Bernstein linearization of rev $p(z)$ is $(\boldsymbol{A}_R, \boldsymbol{B}_R)$. That is, the matrices $\boldsymbol{A}_R$ and $\boldsymbol{B}_R$ have the same form as that of $\boldsymbol{A}$ and $\boldsymbol{B}$, but where $(\boldsymbol{A}, \boldsymbol{B})$ contains c_ks the matrices $(\boldsymbol{A}_R, \boldsymbol{B}_R)$ contains d_ks. To give a new proof that the Bernstein linearization is actually a strong linearization, then, we must find a pair of unimodular matrices $(\boldsymbol{U}, \boldsymbol{V})$ which have $\boldsymbol{U}\boldsymbol{A}_R\boldsymbol{V} = \boldsymbol{B} - \boldsymbol{A}$ and $\boldsymbol{U}\boldsymbol{B}_R\boldsymbol{V} = \boldsymbol{A}$, valid for all choices of coefficients c_k (which determine the corresponding reversed coefficients d_k by the formula above).

Strictly speaking, such a proof is not necessary because the strength of this linearization has been proved elsewhere already, but for surety and because this reversal is unusual here is an outline of the proof.

First, it simplifies matters not to deal with $\boldsymbol{V}$ but rather with $\boldsymbol{V}^{-1}$. Then, our defining conditions become

$$\boldsymbol{U}\boldsymbol{A}_R = (\boldsymbol{B} - \boldsymbol{A})\,\boldsymbol{V}^{-1} \tag{4.3.19}$$

$$\boldsymbol{U}\boldsymbol{B}_R = \boldsymbol{A}\boldsymbol{V}^{-1}\,, \tag{4.3.20}$$

which are linear in the unknowns (the entries of $\boldsymbol{U}$ and of $\boldsymbol{V}^{-1}$). By inspection of the first few dimensions, we find that $\boldsymbol{U}$ and $\boldsymbol{V}^{-1}$ have the following form (using the six-by-six case, for variation, to demonstrate). The anti-diagonal of the general $\boldsymbol{U}$ has entries $-(n-i+1)/i$ for $i = 1, 2, \ldots, n$.

$$\boldsymbol{U} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & -6 \\ 0 & 0 & 0 & 0 & -\frac{5}{2} & u_{2,6} \\ 0 & 0 & 0 & -\frac{4}{3} & u_{3,5} & u_{3,6} \\ 0 & 0 & -\frac{3}{4} & u_{4,4} & u_{4,5} & u_{4,6} \\ 0 & -\frac{2}{5} & u_{5,3} & u_{5,4} & u_{5,5} & u_{5,6} \\ -\frac{1}{6} & u_{6,2} & u_{6,3} & u_{6,4} & u_{6,5} & u_{6,6} \end{bmatrix} \tag{4.3.21}$$

and

$$\boldsymbol{V}^{-1} = \begin{bmatrix} 0 & 0 & 0 & 0 & \Lambda_{1,5} & \Lambda_{1,6} \\ 0 & 0 & 0 & \Lambda_{2,4} & \Lambda_{2,5} & \Lambda_{2,6} \\ 0 & 0 & \Lambda_{3,3} & \Lambda_{3,4} & \Lambda_{3,5} & \Lambda_{3,6} \\ 0 & \Lambda_{4,2} & \Lambda_{4,3} & \Lambda_{4,4} & \Lambda_{4,5} & \Lambda_{4,6} \\ \Lambda_{5,1} & \Lambda_{5,2} & \Lambda_{5,3} & \Lambda_{5,4} & \Lambda_{5,5} & \Lambda_{5,6} \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.3.22}$$

It is an interesting exercise to derive the explicit general formulae

$$u_{i,j} = -\left(\frac{n-i+1}{i}\right)\binom{i}{n+1-j} \qquad 1 \le i, j \le n \tag{4.3.23}$$

$$\Lambda_{i,j} = -\left(\frac{n-i}{j}\right)\binom{i}{n-j} \qquad 1 \le i, j \le n-1 \tag{4.3.24}$$

$$\Lambda_{i,n} = d_i - \frac{n-i}{n}c_n\,, \tag{4.3.25}$$

and to prove that these are not only *necessary* for the equations above, but also *sufficient*. As a quick sketch of how to do it, the matrix $\boldsymbol{B} - \boldsymbol{A}$ is diagonal and gives a direct relationship between the triangular block in $\boldsymbol{V}^{-1}$ and a corresponding portion of $\boldsymbol{U}$; the other equation gives a recurrence relation for the entries of $\boldsymbol{U}$. Comparison of the final columns of the products gives an explicit formula for the final column of $\boldsymbol{V}^{-1}$ and an explicit formula for the entries of $\boldsymbol{U}$ by comparison of the coefficients of the symbols c_k; this formula can be seen to verify the recurrence relation found earlier, closing the circle and establishing sufficiency. Both matrices $\boldsymbol{U}$ and $\boldsymbol{V}$ have determinant ± 1: $\lfloor n/2 \rfloor$ row-permutations brings $\boldsymbol{U}$ to upper triangular form and the determinant $(-1)^n$ (times $(-1)^{\lfloor n/2 \rfloor}$) can be read off as the product of the formerly anti-diagonal elements, and similarly for the upper block of $\boldsymbol{V}^{-1}$ which has one dimension less giving $(-1)^{n-1+\lfloor (n-1)/2 \rfloor}$.

Expansion to the matrix polynomial case merely requires the appropriate tensor product.

4.3.3 The Lagrange interpolational basis

There are by now several Lagrange basis linearizations. The use of barycentric forms means that Lagrange interpolation is efficient and numerically stable and is increasing in popularity [4]. Here is the definition of the first barycentric form for interpolation of polynomials of degree at most ℓ on the $\ell + 1$ distinct nodes $\tau_k \in \mathbb{C}$, $0 \leq k \leq \ell$. Take the partial fraction decomposition of the reciprocal of the node polynomial

$$w(z) = \prod_{k=0}^{\ell} (z - \tau_k) \,, \tag{4.3.26}$$

namely

$$\frac{1}{w(z)} = \sum_{k=0}^{\ell} \frac{\beta_k}{z - \tau_k} \tag{4.3.27}$$

where the coefficients β_k occurring in the partial fraction decomposition are called the *barycentric weights*. A well-known explicit formula for the β_k is

$$\beta_k = \prod_{\substack{j=0 \\ j \neq k}}^{\ell} (\tau_k - \tau_j)^{-1} \,. \tag{4.3.28}$$

The Lagrange basis polynomials are normally written

$$\ell_k(z) = \beta_k \prod_{\substack{j=0 \\ j \neq k}}^{\ell} (z - \tau_j) \,. \tag{4.3.29}$$

For many sets of nodes (Chebyshev nodes on $[-1, 1]$, or roots of unity on the unit disk) the resulting interpolant is also well-conditioned, and can even be "better than optimal" [10], see also [6]. The linearization we use here is "too large" and has (numerically harmless in our experience) spurious roots at infinity[6]; for alternative formulations

[6] This numerical harmlessness needs some explanation. In brief, Lagrange basis matrix polynomial eigenvalues will be well-conditioned only in a compact region determined by the interpolation nodes, and are increasingly ill-conditioned towards infinity; in practice this means only small changes in the data are needed to perturb large finite ill-conditioned eigenvalues out to infinity. Any eigenvalues produced

see [35], [30]. Then the linearization is $z\boldsymbol{C}_1 - \boldsymbol{C}_0$ where

$$\boldsymbol{C}_1 = \begin{bmatrix} 0 & & & & & \\ & 1 & & & & \\ & & 1 & & & \\ & & & 1 & & \\ & & & & 1 & \end{bmatrix} \tag{4.3.30}$$

$$\boldsymbol{C}_0 = \begin{bmatrix} 0 & -\rho_0 & -\rho_1 & -\rho_2 & -\rho_3 & -\rho_4 \\ \beta_0 & \tau_0 & & & & \\ \beta_1 & & \tau_1 & & & \\ \beta_2 & & & \tau_2 & & \\ \beta_3 & & & & \tau_3 & \\ \beta_4 & & & & & \tau_4 \end{bmatrix}. \tag{4.3.31}$$

Matrices $\boldsymbol{E}$ and $\boldsymbol{F}$ demonstrating that this is indeed a linearization can be found in [2] but because we will need them for the Hermite interpolational linearization shortly, we introduce some here:

$$\boldsymbol{E} = \begin{bmatrix} \boldsymbol{I}_n & -\rho\boldsymbol{D}^{-1} \\ 0 & \boldsymbol{I}_{N-n} \end{bmatrix} \tag{4.3.32}$$

and

$$\boldsymbol{F} = \begin{bmatrix} w(z)\boldsymbol{I}_n & 0 \\ \phi(z) \otimes \boldsymbol{I}_n & \boldsymbol{D}^{-1} \end{bmatrix}. \tag{4.3.33}$$

Here $\boldsymbol{D} = \operatorname{diag}(z-\tau_0, z-\tau_1, \ldots, z-\tau_\ell)$, $\phi(z) = [\ell_0(z), \ell_1(z), \ldots, \ell_\ell(z)]^T$ which is also (in the Lagrange case) related to $w(z)$ and the β_k by $\ell_k(z) = \beta_k w(z)/(z-\tau_k)$. Thus $\det \boldsymbol{E} = 1$ and $\det \boldsymbol{F} = 1$ (except at nodes $z = \tau_k$, but we recover the constant by continuity). Multiplying out, we have $\boldsymbol{E}(z\boldsymbol{C}_1 - \boldsymbol{C}_0)\boldsymbol{F} = \operatorname{diag}(\boldsymbol{P}(z), \boldsymbol{I}_n, \ldots, \boldsymbol{I}_n)$. It is amusing that $\boldsymbol{P}(z)$ comes out exactly in the first barycentric form, $\boldsymbol{P}(z) = w(z)\sum_{k=0}^{\ell} \beta_k\rho_k/(z-\tau_k)$.

Then $\det(\tau_k\boldsymbol{C}_1 - \boldsymbol{C}_0) = \det(\rho_k) = \rho_k$ here, $0 \le k \le 4$ and $\deg(z\boldsymbol{C}_1 - \boldsymbol{C}_0) \le 4$. Thus, $p(z) = \det(z\boldsymbol{C}_1 - \boldsymbol{C}_0)$ interpolates the given data, assuming the τ_k are distinct. The $\boldsymbol{X}$ and $\boldsymbol{Y}$ for the standard triple are

$$\boldsymbol{X} = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}, \tag{4.3.34}$$

$$\boldsymbol{Y} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^{\mathrm{T}} \tag{4.3.35}$$

Notice in this case that for the linearization $N = (\ell+2)n$ while $\deg p \le \ell$, and therefore there are at least $2n$ eigenvalues at infinity. This can be inconvenient if n is at all large.

4.3.4 Hermite interpolational basis

The Lagrange linearization of the previous section has been extended to Hermite interpolational bases, where some of the nodes have "flowed together," collapsing to fewer distinct nodes[7].

We suppose that at each remaining distinct node τ_i, $0 \le i \le N-1$, say, there are now $s_i \ge 1$ consecutive pieces of information known, namely $\boldsymbol{P}(\tau_i)$, $\boldsymbol{P}'(\tau_i)/1!$, $\boldsymbol{P}''(\tau_i)/2!$, and so on up to the last one, the value of the $s_i - 1$-th derivative at $z = \tau_i$, namely

numerically that are well outside the region determined by the interpolation nodes are likely easily perturbed all the way to infinity, and can be safely ignored.

[7] A formal definition can be found in [8], for instance. The essential idea is that given two distinct pieces of data $(\tau_k, p(\tau_k))$ and $(\tau_{k+1}, p(\tau_{k+1}))$, we also know the forward difference $(p_{k+1}-p_k)/(\tau_{k+1}-\tau_k)$. In the limit as one node approaches (flows towards) the other, we still know two pieces of information: $p(\tau_k)$ and $p'(\tau_k)$. Hermite interpolation captures this idea.

$\boldsymbol{P}^{(s_i-1)}(\tau_i)/(s_i-1)!$. The integer s_i is called the *confluency* of the node. The known pieces of information are the local Taylor coefficients of the polynomial fitting the data:

$$\rho_{i,j} = \frac{f^{(j)}(\tau_i)}{j!}\,, \quad 0 \le j \le s_i - 1\,. \tag{4.3.36}$$

This gives $1+\ell = \sum s_i$ pieces of information, determining a polynomial of degree at most ℓ. The barycentric weights, this time doubly indexed as $\beta_{i,j}$, are again computed from the partial fraction decomposition of the reciprocal of the node polynomial

$$\frac{1}{w(z)} = \frac{1}{\prod_{i=0}^{N-1}(z-\tau_i)^{s_i}} = \sum_{i=0}^{N-1}\sum_{j=0}^{s_i-1}\frac{\beta_{i,j}}{(z-\tau_i)^{j+1}}\,. \tag{4.3.37}$$

For evaluation of the interpolating polynomial one should use the first or second barycentric form; see [8] for details. For theoretical work with the Hermite interpolational bases, however, we can define

$$H_{i,j}(z) = \sum_{k=0}^{s_i-1-j}\beta_{i,j+k}w(z)(z-\tau_i)^{-k-1}\,. \tag{4.3.38}$$

These polynomials, each of degree at most ℓ, form a basis (a Hermite interpolational basis, to distinguish from the Hermite orthogonal polynomials) for polynomials of degree at most ℓ; moreover they generalize the Lagrange property in that only one Taylor coefficient at only one node is 1 and all the rest are zero.

Note that the derivative $\boldsymbol{P}'(z)$ of a matrix polynomial is a straightforward extension to matrices of the ordinary derivative. It is isomorphic to the matrix with entries that are the ordinary derivatives of the original matrix.

The linearization of the previous section changes to the following elegant form. The matrix $\boldsymbol{C}_1$ is unchanged,

$$\boldsymbol{C}_1 = \begin{bmatrix} 0 & & & & \\ & 1 & & & \\ & & \ddots & & \\ & & & 1 & \\ & & & & 1 \end{bmatrix}, \tag{4.3.39}$$

being $(\ell+2)$ by $(\ell+2)$ as before. The matrix $\boldsymbol{C}_0$ changes, picking up transposed Jordan-like blocks for each distinct node. For instance, suppose we have two distinct nodes, τ_0 and τ_1. Suppose further that τ_0 has confluency $s_0 = 3$ while τ_1 has confluency $s_1 = 2$. This means that we know $f(\tau_0)$, $f'(\tau_0)/1!$, $f''(\tau_0)/2!$, $f(\tau_1)$ and $f'(\tau_1)/1!$. Then,

$$\boldsymbol{C}_0 = \begin{bmatrix} 0 & -f''(\tau_0)/2! & -f'(\tau_0)/1! & -f(\tau_0) & -f'(\tau_1)/1! & -f(\tau_1) \\ \beta_{02} & \tau_0 & & & & \\ \beta_{01} & 1 & \tau_0 & & & \\ \beta_{00} & & 1 & \tau_0 & & \\ \beta_{11} & & & & \tau_1 & \\ \beta_{10} & & & & 1 & \tau_1 \end{bmatrix} \tag{4.3.40}$$

Note the reverse ordering of the derivative values in this formulation.

The matrices $\boldsymbol{E}$ and $\boldsymbol{F}$ demonstrating that this is indeed a linearization have, so far as we know, not been noted in the literature. They are exactly the same as for the Lagrange basis, Equations (4.3.32) and (4.3.33), with appropriately modified meanings

for ϕ and $\boldsymbol{D}$. The new ϕ contains the Hermite interpolational bases in Equation (4.3.38), and now $\boldsymbol{D}$ is not diagonal but rather block diagonal with the transposed Jordan-like blocks above. Both matrices are still unimodular. Again we have $\boldsymbol{E}(z\boldsymbol{C}_1 - \boldsymbol{C}_0)\boldsymbol{F} = \operatorname{diag}(\boldsymbol{P}(z), \boldsymbol{I}_n, \dots, \boldsymbol{I}_n)$.

For the standard triple, take in the scalar case

$$\boldsymbol{Y} = \begin{bmatrix} 1 & 0 & \cdots & 0 \end{bmatrix}^{\mathrm{T}} \tag{4.3.41}$$

but for $\boldsymbol{X}$ take the coefficients of the expansion of the polynomial 1 in this particular Hermite interpolational basis: it is equal to 1 at each node but has all derivatives zero at each node. That is, put

$$\begin{cases} \rho_{ij} = 1 & \text{if } j = 0\,, \\ 0 & \text{otherwise}\,, \end{cases} \tag{4.3.42}$$

and sort them in order:

$$\boldsymbol{X} = \begin{bmatrix} 0 & \rho_{0,s_0-1} & \rho_{0,s_0-2} & \cdots & \rho_{0,0} & \rho_{1,s_1-1} & \cdots & \rho_{n,0} \end{bmatrix}\,. \tag{4.3.43}$$

For the earlier instance (two nodes, of confluency 3 and 2, respectively,

$$\boldsymbol{X} = \begin{bmatrix} 0 & \underbrace{0 & 0 & 1}_{\text{for } \tau_0} & \underbrace{0 & 1}_{\text{for } \tau_1} \end{bmatrix}\,. \tag{4.3.44}$$

Then

$$p^{-1}(z) = \boldsymbol{X}(z\boldsymbol{C}_1 - \boldsymbol{C}_0)^{-1}\boldsymbol{Y}\,. \tag{4.3.45}$$

Remark 4.10. We may re-order the nodes in any fashion we like, and each ordering generates its own linearization (both Hermite and Lagrange). We may also find a linearization where the confluent data is ordered $p(\tau_i)$, $p'(\tau_i)/1!$, $p''(\tau_i)/2!$, etc., although we have not done so.

If there is just one node of confluency ℓ, we recover the standard Frobenius companion (plus two infinite roots):

$$\begin{bmatrix} 0 & & & & \\ 1 & & & & \\ & \ddots & & & \\ & & 1 & & \\ & & & 1 & \end{bmatrix}\,, \quad \begin{bmatrix} 0 & -p_{\ell-1} & -p_{\ell-2} & \cdots & -p_1 & -p_0 \\ 1 & \tau_0 & & & & \\ 0 & 1 & \tau_0 & & & \\ 0 & & 1 & \ddots & & \\ \vdots & & & \ddots & \tau_0 & \\ 0 & & & & 1 & \tau_0 \end{bmatrix}\,. \tag{4.3.46}$$

Here $p_k = p^{(k)}(\tau_0)/k!$ is the ordinary coefficient in the expansion $p(z) = \sum_{k=0}^{\ell} p_k(z-\tau_0)^k$. The numerical stability of these Hermite interpolational linearization has been studied briefly [25] but much remains unknown. We confine ourselves in this paper to the study of the standard triple.

To make a linearization for matrix polynomials out of these scalar linearizations, take the Kronecker tensor product with $\boldsymbol{I}_n$, and insert the appropriate matrix polynomial values and derivative values.

Remark 4.11. The modified linearizations of [35] also have standard triples that can be used for algebraic linearization, and arguably should be tabled here as well. They have the advantage of including fewer eigenvalues at infinity, or no spurious eigenvalues at infinity, which may lead to better algebraic linearizations. However, they are more involved, and we have less numerical experience with them. In particular we do not

understand their dependence on the ordering of the nodes, and so we leave their analysis to a future study.

4.4 Strict Equivalence of Generalized Standard Triples

Theorem 4.12. *If $\phi_k(z)$ for $0 \le k \le \ell$ is one of the degree-graded polynomial bases (e.g. Chebyshev, Newton, Jacobi) then the linearization for a polynomial $p(z)$ expressed in that basis is strictly equivalent to the second linearization for the same polynomial expressed in the monomial basis. That is, there exist unimodular matrices $\boldsymbol{U}$ and $\boldsymbol{V}$ for which $\boldsymbol{C}_{1,m} = \boldsymbol{U}\boldsymbol{C}_{1,\phi}\boldsymbol{V}$ and $\boldsymbol{C}_{0,m} = \boldsymbol{U}\boldsymbol{C}_{0,\phi}\boldsymbol{V}$. The matrix $\boldsymbol{U}$ will depend on the given polynomial $p(z)$. Here the subscript m is short for "monomial".*

Proof. Denote the change-of-bases matrix for polynomials up to degree $\ell - 1$ by $\boldsymbol{\Phi}$. This matrix is ℓ by ℓ. Then we have

$$\begin{bmatrix} \phi_{\ell-1}(z) \\ \phi_{\ell-2}(z) \\ \vdots \\ \phi_1(z) \\ \phi_0(z) \end{bmatrix} = \boldsymbol{\Phi} \begin{bmatrix} z^{\ell-1} \\ \vdots \\ z^2 \\ z \\ 1 \end{bmatrix} . \tag{4.4.1}$$

In all cases considered here, the linearization for a polynomial $p(z)$ of exact degree ℓ has null vectors of the form

$$\boldsymbol{N} = \begin{bmatrix} \phi_{\ell-1}(\lambda) \\ \phi_{\ell-2}(\lambda) \\ \vdots \\ \phi_1(\lambda) \\ \phi_0(\lambda) \end{bmatrix} \tag{4.4.2}$$

where λ is a root of $p(z)$. That is,

$$(\lambda \boldsymbol{C}_{1,\phi} - \boldsymbol{C}_{0,\phi})\, \boldsymbol{N} = \boldsymbol{0} . \tag{4.4.3}$$

Using the $\boldsymbol{\Phi}$ formula above, we have $\boldsymbol{V} = \boldsymbol{\Phi}$. By direct computation, we find that $\boldsymbol{U} = \boldsymbol{C}_{0,m}\boldsymbol{\Phi}^{-1}\boldsymbol{C}_{0,\phi}^{-1}$ necessarily giving $\boldsymbol{U}\boldsymbol{C}_{0,\phi}\boldsymbol{V} = \boldsymbol{C}_{0,m}$. Since $\boldsymbol{V} = \boldsymbol{\Phi}$ is unimodular, all that remains is to show that $\boldsymbol{U}$ is unimodular, and that it satisfies $\boldsymbol{U}\boldsymbol{C}_{1,\phi}\boldsymbol{V} = \boldsymbol{C}_{1,m}$ as well. Since $\boldsymbol{C}_{1,m}$ (which corresponds to the monomial basis) is the identity matrix except for the $1,1$ entry which is $a_\ell \neq 0$, the leading coefficient of the polynomial, this last is straightforward. Indeed, the action of premultiplying by $\boldsymbol{C}_{0,m}$ and postmultiplying by $\boldsymbol{C}_{0,m}^{-1}$ cancels the coefficient a_ℓ in the $1,1$ entry of $\boldsymbol{U}$, and thus by continuity this construction works even if $a_\ell = 0$, that is for polynomials of *grade* ℓ. Then since $\boldsymbol{\Phi}$ is upper triangular for degree-graded matrices and lower triangular for Bernstein matrices, and $\boldsymbol{U}$ has in that case zeros in the first column below the diagonal entry, $\boldsymbol{U}$ is unimodular.

To make this work for regular matrix polynomials of dimension n, we must use the tensor product $\boldsymbol{\Phi} \otimes \boldsymbol{I}$. □

We illustrate this proof with a four by four example in the Bernstein basis, which has somewhat contrasting behaviour. If $p(z) = a_0 B_0^4(z) + a_1 B_1^4(z) + a_2 B_2^4(z) + a_3 B_3^4(z) +$

$a_4 B_4^4(z)$, then

$$\boldsymbol{U} = \begin{bmatrix} 1 & \frac{1}{4}\,a_0 - a_1 + \frac{3}{2}\,a_2 & -\frac{1}{6}\,a_0 + \frac{2}{3}\,a_1 & \frac{1}{4}\,a_0 \\ 0 & 1/4 & 0 & 0 \\ 0 & 1/4 & 1/6 & 0 \\ 0 & 1/4 & 1/3 & 1/4 \end{bmatrix} \tag{4.4.4}$$

and

$$\boldsymbol{V} = \begin{bmatrix} 4 & 0 & 0 & 0 \\ -6 & 6 & 0 & 0 \\ 4 & -8 & 4 & 0 \\ -1 & 3 & -3 & 1 \end{bmatrix} . \tag{4.4.5}$$

Direct computation shows that both matrices are nonsingular irrespective of the values of the a_k, and that these transform the Bernstein linearization

$$\boldsymbol{C}_{0,\text{Bernstein}} = \begin{bmatrix} -a_3 & -a_2 & -a_1 & -a_0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{4.4.6}$$

and

$$\boldsymbol{C}_{1,\text{Bernstein}} = \begin{bmatrix} \frac{a_4}{4} - a_3 & -a_2 & -a_1 & -a_0 \\ 1 & 2/3 & 0 & 0 \\ 0 & 1 & 3/2 & 0 \\ 0 & 0 & 1 & 4 \end{bmatrix} \tag{4.4.7}$$

to

$$\boldsymbol{C}_{0,\text{monomial}} = \begin{bmatrix} -b_3 & -b_2 & -b_1 & -b_0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{4.4.8}$$

with $-b_3 = 4\,a_0 - 12\,a_1 + 12\,a_2 - 4\,a_3$, $-b_2 = -6\,a_0 + 12\,a_1 - 6\,a_2$, $-b_1 = 4\,a_0 - 4\,a_1$, and $-b_0 = -a_0$. Also, $\boldsymbol{U}\boldsymbol{C}_{1,\text{Bernstein}}\boldsymbol{V}$ becomes the identity matrix except the $1,1$ entry is $b_4 = a_0 - 4\,a_1 + 6\,a_2 - 4\,a_3 + a_4$. These are the correct coefficients of the same polynomial expressed in the monomial basis.

Now, the vector $\boldsymbol{N}$ used in the second proof of Theorem 4.8 is not here composed of Bernstein basis elements up to the index $\ell - 1$, but rather is composed of *multiples* of Bernstein basis elements of *order* $\ell - 1$, which happen to be some of the Bernstein basis elements of order ℓ, divided by $1 - z$. Here, where the Bernstein basis of order three gives elements z^3, $3z^2\,(1-z)$, $3z\,(1-z)^2$, $(1-z)^3$,

$$\boldsymbol{N} = \begin{bmatrix} 4\,z^3 \\ 6\,z^2\,(1-z) \\ 4\,z\,(1-z)^2 \\ (1-z)^3 \end{bmatrix} . \tag{4.4.9}$$

The ratios of these to the lower-degree Bernstein basis are 4/1, 4/2, 4/3, and 4/4. This gives the detail needed for the proof of our general main theorem, Theorem 4.8, in the case of Bernstein bases.

4.4.1 The Lagrange interpolational case

As is usual we denote a Lagrange basis element on the distinct nodes $[\tau_0, \tau_1, \ldots, \tau_\ell]$ by $\ell_k(z) = \beta_k \prod_{j \neq k}(z - \tau_j)$. The use of the symbol ℓ by itself denotes an integer, namely the grade of the polynomial; the distinction is that ℓ with a subscript and a variable $\ell_j(z)$ denotes a Lagrange basis polynomial. This should not cause confusion.

Remark 4.13. Many people think of "interpolation" as *meaning* the construction of a monomial basis polynomial $p(z) = a_\ell z^\ell + \cdots + a_0$ that fits the given data $p(\tau_k) = \rho_k$ for $0 \leq k \leq \ell$. This is naive. Interpolation truly means constructing a polynomial in any basis that we may use to evaluate $p(z)$ for z different to the values at the nodes. A very stable and convenient way to do this is by the barycentric form of Lagrange interpolants [4]. Constructing an interpolant in a Newton basis by using divided differences or the monomial basis by using a Vandermonde matrix is changing the basis. Changing bases can have condition number exponential in the degree, and is usually a bad idea. In practice, we use the barycentric form [4]. For the purposes of proof of equivalence, we here occasionally use the Vandermonde matrix, and we think about the explicit construction of the monomial basis. This is not used in numerical practice.

Theorem 4.14. *If $\phi_k(z)$ for $0 \leq k \leq \ell$ is a Lagrange basis on distinct nodes τ_0, τ_1, ..., τ_ℓ, then the $\ell + 2$ by $\ell + 2$ linearization for a polynomial $p(z)$ expressed in that basis is strictly equivalent to what is called the second standard linearization in [17] for same polynomial expressed in the monomial basis but regarded as having grade $\ell+2$ (i.e. with zero coefficients padding the terms $z^{\ell+2}$ and $z^{\ell+1}$. That is, there exist unimodular matrices $\boldsymbol{U}$ and $\boldsymbol{V}$ for which $\boldsymbol{C}_{1,\phi} = \boldsymbol{U}\boldsymbol{C}_{1,m}\boldsymbol{V}$ and $\boldsymbol{C}_{0,\phi} = \boldsymbol{U}\boldsymbol{C}_{0,m}\boldsymbol{V}$, where now the matrices in the second standard linearization have dimension larger by two than needed for the exact degree. The matrix $\boldsymbol{U}$ will depend on the given polynomial $p(z)$.*

Proof. For the Lagrange basis linearization, the right null vector is not of the form indicated in the proof of Theorem 4.8 but rather of the form

$$\boldsymbol{N} = \begin{bmatrix} w(\lambda) \\ \ell_0(\lambda) \\ \ell_1(\lambda) \\ \vdots \\ \ell_\ell(\lambda) \end{bmatrix} . \tag{4.4.10}$$

Here $w(z) = \prod_{k=0}^{\ell}(z - \tau_k)$ is of degree $\ell + 1$ and all the other entries, being elements of the Lagrange basis on $\ell+1$ nodes, are of degree ℓ. Thus $\boldsymbol{\Phi}$ is dimension $\ell+2$ by $\ell+2$ and has first column e_1; that is, 1 in the first entry and zeros below it. The rest of the first row contains the coefficients of $w(z)$ expanded in the monomial basis. The remaining rows of $\boldsymbol{\Phi}$ contain the coefficients of the monomial expansions of the Lagrange basis polynomial; that is, the inverse of the transposed Vandermonde matrix, which relates

the Lagrange interpolation basis to the monomial basis. Call that block $\hat{\boldsymbol{\Phi}}$. Explicitly,

$$\hat{\boldsymbol{\Phi}}^{-1} = \begin{bmatrix} \tau_0^\ell & \tau_1^\ell & \cdots & \tau_\ell^\ell \\ \tau_0^{\ell-1} & \tau_1^{\ell-1} & \cdots & \tau_\ell^{\ell-1} \\ \vdots & & & \vdots \\ \tau_0 & \tau_1 & \cdots & \tau_\ell \\ 1 & 1 & \cdots & 1 \end{bmatrix} . \tag{4.4.11}$$

Then $\boldsymbol{U}$ is straightforwardly seen to be $\operatorname{diag}(1, \hat{\boldsymbol{\Phi}}^{-1})$ and as in the degree-graded case $\boldsymbol{V} = \boldsymbol{\Phi}$. Further, $\boldsymbol{U}\boldsymbol{C}_{0,\phi}\boldsymbol{V}$ has as its first row $[0, -\rho\hat{\boldsymbol{\Phi}}]$. But $\rho\hat{\boldsymbol{\Phi}}$ is by the Vandermonde matrix simply the negative of the vector of monomial coefficients, $-[0, a_{\ell-1}, a_{\ell-2}, \ldots, a_0]$. More, the block underneath, namely $\hat{\boldsymbol{\Phi}}^{-1}\operatorname{diag}(\tau_0, \tau_1, \ldots, \tau_\ell)\hat{\boldsymbol{\Phi}}$ turns out to be simple, because

$$\hat{\boldsymbol{\Phi}}^{-1}\operatorname{diag}(\tau_0, \tau_1, \ldots, \tau_\ell) = \begin{bmatrix} \tau_0^{\ell+1} & \tau_1^{\ell+1} & \cdots & \tau_\ell^{\ell+1} \\ \tau_0^\ell & \tau_1^\ell & \cdots & \tau_\ell^\ell \\ \vdots & & & \vdots \\ \tau_0^2 & \tau_1^2 & \cdots & \tau_\ell^2 \\ \tau_0 & \tau_1 & \cdots & \tau_\ell \end{bmatrix} \tag{4.4.12}$$

Multiplying this by $\hat{\boldsymbol{\Phi}}$ shifts the identity matrix down one diagonal, giving the correct form for the second standard linearization matrix $\boldsymbol{C}_{0,m}$.

As in the previous theorem, to construct $\boldsymbol{U}$ and $\boldsymbol{V}$ for n-dimensional regular matrix polynomials, we must take the tensor product $\boldsymbol{\Phi} \otimes \boldsymbol{I}$. □

Again we illustrate this proof with an example, this time of interpolation at the four points $[-1, -1/2, 1/2, 1]$. This will give rise to a polynomial of degree at most 3. If the values this polynomial takes at these four points are ρ_0, ρ_1, ρ_2, and ρ_3, then the equivalent polynomial expressed in the monomial basis has coefficients

$$\begin{aligned} a_0 &= -1/6\,\rho_0 + 2/3\,\rho_1 + 2/3\,\rho_2 - 1/6\,\rho_3 \\ a_1 &= 1/6\,\rho_0 - 4/3\,\rho_1 + 4/3\,\rho_2 - 1/6\,\rho_3 \\ a_2 &= 2/3\,\rho_0 - 2/3\,\rho_1 - 2/3\,\rho_2 + 2/3\,\rho_3 \\ a_3 &= 2/3\,\rho_3 - 4/3\,\rho_2 + 4/3\,\rho_1 - 2/3\,\rho_0 . \end{aligned} \tag{4.4.13}$$

Expressing this as a polynomial of grade 5, that is $p(z) = 0 \cdot z^5 + 0 \cdot z^4 + a_3 z^3 + a_2 z^2 + a_1 z + a_0$, we get the second standard linearization

$$\boldsymbol{C}_{0,\text{monomial}} = \begin{bmatrix} 0 & -a_3 & -a_2 & -a_1 & -a_0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} \tag{4.4.14}$$

and

$$\boldsymbol{C}_{1,\text{monomial}} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}. \tag{4.4.15}$$

The Lagrange basis linearization of [10], which admittedly has two extra infinite eigenvalues, is

$$\boldsymbol{C}_{0,\text{Lagrange}} = \begin{bmatrix} 0 & -\rho_3 & -\rho_2 & -\rho_1 & -\rho_0 \\ 2/3 & 1 & 0 & 0 & 0 \\ -4/3 & 0 & 1/2 & 0 & 0 \\ 4/3 & 0 & 0 & -1/2 & 0 \\ -2/3 & 0 & 0 & 0 & -1 \end{bmatrix} \tag{4.4.16}$$

and

$$\boldsymbol{C}_{1,\text{Lagrange}} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}. \tag{4.4.17}$$

For these interpolation nodes, direct computation shows

$$\boldsymbol{U} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1/8 & -1/8 & -1 \\ 0 & 1 & 1/4 & 1/4 & 1 \\ 0 & 1 & 1/2 & -1/2 & -1 \\ 0 & 1 & 1 & 1 & 1 \end{bmatrix} \tag{4.4.18}$$

and

$$\boldsymbol{V} = \begin{bmatrix} 1 & 0 & -5/4 & 0 & 1/4 \\ 0 & 2/3 & 2/3 & -1/6 & -1/6 \\ 0 & -4/3 & -2/3 & 4/3 & 2/3 \\ 0 & 4/3 & -2/3 & -4/3 & 2/3 \\ 0 & -2/3 & 2/3 & 1/6 & -1/6 \end{bmatrix}. \tag{4.4.19}$$

4.5 Schur complement-based proofs

In Section 4.2 we gave two short universal proofs of all the theorems in this section. Each individual proof in this section is therefore redundant. We include them here both for surety (thus giving a third proof of each theorem) and because they give insight and may be relevant to any numerical analysis. We will use the Schur Complement, in the

following form: assuming a matrix $\boldsymbol{R}$ is partitioned into

$$\boldsymbol{R} = \begin{bmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{C} & \boldsymbol{D} \end{bmatrix} \tag{4.5.1}$$

where $\boldsymbol{A} \in \mathbb{C}^{r\times r}$, $\boldsymbol{B} \in \mathbb{C}^{r\times(N-r)}$, $\boldsymbol{C} \in \mathbb{C}^{(N-r)\times r}$ and $\boldsymbol{D} \in \mathbb{C}^{(N-r)\times(N-r)}$ is assumed invertible, then

$$\boldsymbol{R} = \begin{bmatrix} \boldsymbol{I}_n & \boldsymbol{B}\boldsymbol{D}^{-1} \\ 0 & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{A} - \boldsymbol{B}\boldsymbol{D}^{-1}\boldsymbol{C} & 0 \\ \boldsymbol{C} & \boldsymbol{D} \end{bmatrix}. \tag{4.5.2}$$

If further the Schur Complement $\boldsymbol{A} - \boldsymbol{B}\boldsymbol{D}^{-1}\boldsymbol{C}$ is invertible, then

$$\boldsymbol{R}^{-1} = \begin{bmatrix} (\boldsymbol{A} - \boldsymbol{B}\boldsymbol{D}^{-1}\boldsymbol{C})^{-1} & -(\boldsymbol{A} - \boldsymbol{B}\boldsymbol{D}^{-1}\boldsymbol{C})^{-1}\boldsymbol{B}\boldsymbol{D}^{-1} \\ -\boldsymbol{D}^{-1}\boldsymbol{C}(\boldsymbol{A} - \boldsymbol{B}\boldsymbol{D}^{-1}\boldsymbol{C})^{-1} & \boldsymbol{D}^{-1} + \boldsymbol{D}^{-1}\boldsymbol{C}(\boldsymbol{A} - \boldsymbol{B}\boldsymbol{D}^{-1}\boldsymbol{C})^{-1}\boldsymbol{B}\boldsymbol{D}^{-1} \end{bmatrix} \tag{4.5.3}$$

as can be verified by block multiplication by $\boldsymbol{R}$. We will use $\boldsymbol{S}$ for the Schur Complement $\boldsymbol{S} = \boldsymbol{A} - \boldsymbol{B}\boldsymbol{D}^{-1}\boldsymbol{C}$. We will take $\boldsymbol{R} = \boldsymbol{L}(z) = z\boldsymbol{C}_1 - \boldsymbol{C}_0$. We may already use this to establish for each of the four classes of linearizations that

$$\det \boldsymbol{R} = \det(z\boldsymbol{C}_1 - \boldsymbol{C}_0) = \det(\boldsymbol{A} - \boldsymbol{B}\boldsymbol{D}^{-1}\boldsymbol{C}) \det \boldsymbol{D} = \det \boldsymbol{P}(z)\,. \tag{4.5.4}$$

Notice that the coefficients of $\boldsymbol{P}$ do not appear in the $\boldsymbol{D}$ block (in any of our linearizations). Thus the Schur Complement carries all the information particular to $\boldsymbol{P}(z)$. The computations verifying (4.5.4) are not obvious but in each case $\boldsymbol{D}^{-1}$ plays an important role. We will see that generically $\boldsymbol{D}^{-1}$ exists, except for isolated values of z, which we can safely ignore and recover later by continuity.
We take each case in turn.

Theorem 4.15. *If* $\boldsymbol{C}_1 = \operatorname{diag}\left[\dfrac{1}{\alpha_{\ell-1}}\boldsymbol{P}_\ell \quad \boldsymbol{I}_n \quad \boldsymbol{I}_n \quad \cdots \quad \boldsymbol{I}_n\right]$ *and*

$$\boldsymbol{C}_0 = \left[\begin{array}{c|ccccc} \dfrac{\beta_{\ell-1}}{\alpha_{\ell-1}}\boldsymbol{P}_\ell - \boldsymbol{P}_{\ell-1} & \dfrac{\gamma_{\ell-1}}{\alpha_{\ell-1}}\boldsymbol{P}_\ell - \boldsymbol{P}_{\ell-2} & -\boldsymbol{P}_{\ell-1} & \cdots & -\boldsymbol{P}_0 \\ \hline \alpha_{\ell-2}\boldsymbol{I}_\ell & \beta_{\ell-2}\boldsymbol{I}_n & \gamma_{\ell-2}\boldsymbol{I}_n & & \\ & \alpha_{\ell-3}\boldsymbol{I}_n & \beta_{\ell-3}\boldsymbol{I}_n & \gamma_{\ell-3}\boldsymbol{I}_n & \\ & & \ddots & \ddots & \gamma_1\boldsymbol{I}_n \\ & & & \alpha_0\boldsymbol{I}_n & \beta_0\boldsymbol{I}_n \end{array}\right] \tag{4.5.5}$$

and $\boldsymbol{X} = \begin{bmatrix} 0 & 0 & \cdots & 0 & \boldsymbol{I}_n \end{bmatrix}$ *and* $\boldsymbol{Y} = \begin{bmatrix} \boldsymbol{I}_n & 0 & 0 & \cdots & 0 \end{bmatrix}$ *then* $\boldsymbol{X}(z\boldsymbol{C}_1 - \boldsymbol{C}_0)^{-1}\boldsymbol{Y} = \boldsymbol{P}^{-1}(z)$ *where* $\boldsymbol{P}(z) = \sum_{k=0}^{\ell} \boldsymbol{P}_k\phi_k(z)$ *except for such* z *that* $\det \boldsymbol{P}(z) = 0$. *As in Section 4.3.1 the polynomials* $\phi_k(z)$ *satisfy* $z\phi_k = \alpha_k\phi_{k+1} + \beta_k\phi_k + \gamma_k\phi_{k-1}$, $\phi_{-1} = 0$, $\phi_0 = 1$, $\phi_1 = {}^{(z-\beta_0)}/_{\alpha_0}$. *In this theorem,* $\ell \geq 2$ *and* $N = \ell n$, *and if* $\boldsymbol{P}_\ell \neq 0_n$ *then degree* $\boldsymbol{P} = \ell$.

That this is a linearization is well-known; see e.g. [3]. We only prove $\boldsymbol{P}^{-1}(z) = \boldsymbol{X}\boldsymbol{R}^{-1}\boldsymbol{Y}$, here.

Proof. We use the first block column of Schur Complement inverse formula

$$\boldsymbol{R}^{-1} = \begin{bmatrix} \boldsymbol{S}^{-1} & * \\ -\boldsymbol{D}^{-1}\boldsymbol{C}\boldsymbol{S}^{-1} & * \end{bmatrix}. \tag{4.5.6}$$

Here

$$\boldsymbol{D} = \begin{bmatrix} (z-\beta_{\ell-2})\boldsymbol{I}_n & -\gamma_{\ell-2}\boldsymbol{I}_n & & & \\ -\alpha_{\ell-3}\boldsymbol{I}_n & (z-\beta_{\ell-3})\boldsymbol{I}_n & -\gamma_{\ell-3}\boldsymbol{I}_n & & \\ & -\alpha_{\ell-4}\boldsymbol{I}_n & & \ddots & \\ & & \ddots & \ddots & -\gamma_1\boldsymbol{I}_n \\ & & & -\alpha_0\boldsymbol{I}_n & (z-\beta_0)\boldsymbol{I} \end{bmatrix} \tag{4.5.7}$$

is block tridiagonal, and

$$\boldsymbol{C} = \begin{bmatrix} -\alpha_{\ell-2}\boldsymbol{I}_n \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} . \tag{4.5.8}$$

By inspection $\boldsymbol{V} = -\boldsymbol{D}^{-1}\boldsymbol{C}$ is

$$\boldsymbol{V} = q \begin{bmatrix} \phi_{\ell-2}(z)\boldsymbol{I}_n \\ \vdots \\ \phi_2(z)\boldsymbol{I}_n \\ \phi_1(z)\boldsymbol{I}_n \\ \phi_0(z)\boldsymbol{I}_n \end{bmatrix} \tag{4.5.9}$$

for some constant q, because

$$-\alpha_k\phi_{k+1}(z) + (z-\beta_k)\phi_k(z) - \gamma_k\phi_{k-1}(z) = 0 \tag{4.5.10}$$

for $k = 0, 1, \cdots, \ell-3$. The constant q is obtained from

$$q \cdot (z-\beta_{\ell-2})\phi_{\ell-2}(z) - q \cdot \gamma_{\ell-2}\phi_{\ell-3}(z) = +\alpha_{\ell-2} \tag{4.5.11}$$

or

$$q \cdot [\phi_{\ell-1}(z)] = +1 \tag{4.5.12}$$

So

$$q = \frac{+1}{\phi_{\ell-1}(z)} . \tag{4.5.13}$$

It follows that

$$\boldsymbol{S} = \frac{z-\beta_{\ell-1}}{\alpha_{\ell-1}}\boldsymbol{P}_\ell + \boldsymbol{P}_{\ell-1} + \begin{bmatrix} \dfrac{-\gamma_{\ell-1}}{\alpha_{\ell-1}}\boldsymbol{P}_\ell + \boldsymbol{P}_{\ell-2} & \boldsymbol{P}_{\ell-3} & \cdots & \boldsymbol{P}_0 \end{bmatrix} \begin{bmatrix} \phi_{\ell-2}(z) \\ \phi_{\ell-3}(z) \\ \vdots \\ \phi_0(z) \end{bmatrix} \cdot \frac{1}{\phi_{\ell-1}(z)}$$

$$= \frac{\dfrac{z-\beta_{\ell-1}}{\alpha_{\ell-1}}\phi_{\ell-1}(z)\boldsymbol{P}_\ell + \phi_{\ell-1}(z)\boldsymbol{P}_{\ell-1} - \dfrac{\gamma_{\ell-1}}{\alpha_{\ell-1}}\phi_{\ell-2}(z)\boldsymbol{P}_\ell + \phi_{\ell-2}(z)\boldsymbol{P}_{\ell-2} + \cdots + \phi_0(z)\boldsymbol{P}_0}{\phi_{\ell-1}(z)}$$

$$= \frac{\sum_{k=0}^{\ell}\phi_k(z)\boldsymbol{P}_k}{\phi_{\ell-1}(z)} = \frac{\boldsymbol{P}(z)}{\phi_{\ell-1}(z)}\,. \tag{4.5.14}$$

Thus

$$-\boldsymbol{D}^{-1}\boldsymbol{C}\boldsymbol{S}^{-1} = \begin{bmatrix} \phi_{\ell-2}(z)\boldsymbol{I}_n \\ \vdots \\ \phi_0(z)\boldsymbol{I}_n \end{bmatrix} \boldsymbol{P}^{-1}(z) \tag{4.5.15}$$

because $\dfrac{1}{\phi_{\ell-1}(z)}\boldsymbol{S}^{-1} = \boldsymbol{P}^{-1}(z)$. Finally, $\phi_0(z) = 1$, so the bottom block is $\boldsymbol{P}^{-1}(z)$, establishing that

$$\boldsymbol{X} = \begin{bmatrix} 0 & 0 & \cdots & 0 & \boldsymbol{I}_n \end{bmatrix} \tag{4.5.16}$$

$$\boldsymbol{Y} = \begin{bmatrix} \boldsymbol{I}_n & 0 & \cdots & 0 & 0 \end{bmatrix}^T \tag{4.5.17}$$

will produce $\boldsymbol{X}\boldsymbol{R}^{-1}\boldsymbol{Y} = \boldsymbol{P}^{-1}(z)$. □

Theorem 4.16. *Put*

$$\boldsymbol{C}_1 = \begin{bmatrix} \dfrac{1}{\ell}\boldsymbol{P}_\ell - \boldsymbol{P}_{\ell-1} & -\boldsymbol{P}_{\ell-2} & \cdots & -\boldsymbol{P}_1 & -\boldsymbol{P}_0 \\ \boldsymbol{I}_n & \dfrac{2}{\ell-1}\boldsymbol{I}_n & & & \\ & \boldsymbol{I}_n & \dfrac{3}{\ell-2}\boldsymbol{I}_n & & \\ & & \ddots & \ddots & \\ & & & \boldsymbol{I}_n & \dfrac{\ell}{1}\boldsymbol{I}_n \end{bmatrix} \tag{4.5.18}$$

and

$$\boldsymbol{C}_0 = \begin{bmatrix} -\boldsymbol{P}_{\ell-1} & -\boldsymbol{P}_{\ell-2} & \cdots & -\boldsymbol{P}_1 & -\boldsymbol{P}_0 \\ \boldsymbol{I}_n & 0 & & & \\ & \boldsymbol{I}_n & 0 & & \\ & & \ddots & \ddots & \\ & & & \boldsymbol{I}_n & 0 \end{bmatrix} \tag{4.5.19}$$

and $\boldsymbol{Y} = \begin{bmatrix}\boldsymbol{I}_n & 0 & \cdots & 0 & 0\end{bmatrix}^T$ *with* $\boldsymbol{X} = \begin{bmatrix}\frac{1}{\ell}\boldsymbol{I}_n & \frac{2}{\ell}\boldsymbol{I}_n & \frac{3}{\ell}\boldsymbol{I}_n & \cdots & \frac{\ell}{\ell}\boldsymbol{I}_n\end{bmatrix}$. *Then* $\boldsymbol{X}(z\boldsymbol{C}_1 - \boldsymbol{C}_0)^{-1}\boldsymbol{Y} = \boldsymbol{P}^{-1}(z)$, *unless* $z \in \Lambda(\boldsymbol{P})$, *and* $\det \boldsymbol{P}(z) = \det \boldsymbol{R}(z) = \det(z\boldsymbol{C}_1 - \boldsymbol{C}_0)$.

Proof. This linearization is proved e.g. in [29], but for convenience we supply one here as well. The Schur factoring is

$$\boldsymbol{R} = \begin{bmatrix}\boldsymbol{I}_n & \boldsymbol{B}\boldsymbol{D}^{-1} \\ 0 & \boldsymbol{I}_{N-r}\end{bmatrix}\begin{bmatrix}\boldsymbol{S} & 0 \\ \boldsymbol{C} & \boldsymbol{D}\end{bmatrix} \tag{4.5.20}$$

where $\boldsymbol{S} = \boldsymbol{A} - \boldsymbol{B}\boldsymbol{D}^{-1}\boldsymbol{C}$ is the Schur Complement. Here

$$A = \frac{z}{\ell}\boldsymbol{P}_\ell + (1-z)\boldsymbol{P}_{\ell-1} \tag{4.5.21}$$

$$\boldsymbol{B} = \begin{bmatrix}(1-z)\boldsymbol{P}_{\ell-2} & (1-z)\boldsymbol{P}_{\ell-3} & \cdots & (1-z)\boldsymbol{P}_0\end{bmatrix} \tag{4.5.22}$$

$$\boldsymbol{C} = \begin{bmatrix}(z-1)\boldsymbol{I}_n \\ 0 \\ 0 \\ \vdots \\ 0\end{bmatrix} \tag{4.5.23}$$

and

$$\boldsymbol{D} = \begin{bmatrix}\frac{2}{\ell-1}z\boldsymbol{I}_n & & & & \\ (z-1)\boldsymbol{I}_n & \frac{3}{\ell-2}z\boldsymbol{I}_n & & & \\ & (z-1)\boldsymbol{I}_n & \ddots & & \\ & & & \ddots & \\ & & & (z-1)\boldsymbol{I}_n & \frac{\ell}{1}z\boldsymbol{I}_n\end{bmatrix} \tag{4.5.24}$$

Therefore $\boldsymbol{V} = \boldsymbol{D}^{-1}\boldsymbol{C}$ satisfies

$$\begin{bmatrix}\frac{2}{\ell-1}z\boldsymbol{I}_n & & & & \\ (z-1)\boldsymbol{I}_n & \frac{3}{\ell-2}z\boldsymbol{I}_n & & & \\ & (z-1)\boldsymbol{I}_n & \frac{4}{\ell-3}z\boldsymbol{I}_n & & \\ & & \ddots & & \\ & & & \ddots & \\ & & & (z-1)\boldsymbol{I}_n & \frac{\ell}{1}z\boldsymbol{I}_n\end{bmatrix}\begin{bmatrix}\boldsymbol{v}_1 \\ \boldsymbol{v}_2 \\ \vdots \\ \boldsymbol{v}_{\ell-1}\end{bmatrix} = \begin{bmatrix}(z-1)\boldsymbol{I}_n \\ 0 \\ \vdots \\ 0\end{bmatrix} \tag{4.5.25}$$

So

$$\boldsymbol{v}_1 = \frac{\ell-1}{2}\left(\frac{z-1}{z}\right)\boldsymbol{I}_n = -\frac{\ell-1}{2}\left(\frac{1-z}{z}\right)\boldsymbol{I}_n \tag{4.5.26}$$

$$\boldsymbol{v}_2 = -\frac{\ell-2}{3}\cdot\boldsymbol{v}_1 = -\frac{\ell-2}{3}\cdot\frac{\ell-1}{2}\cdot\left(\frac{1-z}{z}\right)^2\boldsymbol{I}_n \tag{4.5.27}$$

$$\boldsymbol{v}_3 = -\frac{\ell-3}{4}\cdot\frac{\ell-2}{3}\cdot\frac{\ell-1}{2}\left(\frac{1-z}{z}\right)^3\boldsymbol{I}_n \tag{4.5.28}$$

and so on; by inspection, confirmed by a formal induction not given here,

$$\boldsymbol{v}_k = -\frac{(\ell-1)!}{(\ell-k-1)!(k+1)!}\left(\frac{1-z}{z}\right)^k\boldsymbol{I}_n = -\frac{1}{\ell}\binom{\ell}{k+1}\left(\frac{1-z}{z}\right)^k\boldsymbol{I}_n \tag{4.5.29}$$

for $k = 1, \cdots, \ell-1$. Thus

$$\begin{aligned}\boldsymbol{S} &= \frac{z}{\ell}\boldsymbol{P}_\ell + (1-z)\boldsymbol{P}_{\ell-1} + (1-z)\begin{bmatrix}\boldsymbol{P}_{\ell-2} & \boldsymbol{P}_{\ell-3} & \cdots & \boldsymbol{P}_0\end{bmatrix}\begin{bmatrix}\frac{1}{\ell}\binom{\ell}{2}\left(\frac{1-z}{z}\right)\boldsymbol{I}_n\\ \frac{1}{\ell}\binom{\ell}{3}\left(\frac{1-z}{z}\right)^2\boldsymbol{I}_n\\ \vdots\\ \frac{1}{\ell}\binom{\ell}{\ell}\left(\frac{1-z}{z}\right)^{\ell-1}\boldsymbol{I}_n\end{bmatrix}\\ &= \frac{1}{\ell z^{\ell-1}}\cdot\left[z^\ell\boldsymbol{P}_\ell + \ell z^{\ell-1}(1-z)\boldsymbol{P}_{\ell-1} + \binom{\ell}{2}z^{\ell-2}(1-z)^2\boldsymbol{P}_{\ell-2} + \cdots + \binom{\ell}{\ell}(1-z)^\ell\boldsymbol{P}_0\right]\\ &= \frac{\boldsymbol{P}(z)}{\ell z^{\ell-1}}\,.\end{aligned} \tag{4.5.30}$$

Moreover,

$$\boldsymbol{S}^{-1} = \ell z^{\ell-1}\boldsymbol{P}^{-1}(z) \tag{4.5.31}$$

and the first column of $\boldsymbol{R}^{-1}$ is

$$\begin{bmatrix}\boldsymbol{S}^{-1}\\ -\boldsymbol{D}^{-1}\boldsymbol{C}\boldsymbol{S}^{-1}\end{bmatrix} = \begin{bmatrix}\ell z^{\ell-1}\boldsymbol{P}^{-1}\\ \ell z^{\ell-1}\cdot\frac{1}{\ell}\binom{\ell}{2}\left(\frac{1-z}{z}\right)\boldsymbol{P}^{-1}\\ \ell z^{\ell-1}\cdot\frac{1}{\ell}\binom{\ell}{3}\left(\frac{1-z}{z}\right)^2\boldsymbol{P}^{-1}\\ \ell z^{\ell-1}\cdot\frac{1}{\ell}\binom{\ell}{4}\left(\frac{1-z}{z}\right)^3\boldsymbol{P}^{-1}\\ \vdots\\ \ell z^{\ell-1}\cdot\frac{1}{\ell}\binom{\ell}{\ell}\left(\frac{1-z}{z}\right)^{\ell-1}\boldsymbol{P}^{-1}\end{bmatrix} = \begin{bmatrix}nz^{n-1}\boldsymbol{P}^{-1}\\ \binom{\ell}{2}z^{\ell-2}(1-z)\boldsymbol{P}^{-1}\\ \binom{\ell}{3}z^{\ell-3}(1-z)^2\boldsymbol{P}^{-1}\\ \vdots\\ \binom{\ell}{\ell}z^0(1-z)^{\ell-1}\boldsymbol{P}^{-1}\end{bmatrix} \tag{4.5.32}$$

We now notice that 1, expressed as a linear combination of

$$\binom{\ell}{1}z^{\ell-1}, \binom{\ell}{2}z^{\ell-2}(1-z), \cdots, \binom{\ell}{\ell}z^0(1-z)^{\ell-1} \tag{4.5.33}$$

(these are the elements of the null vector used in the proof of Theorem 4.8) is

$$\begin{aligned}1 &= \frac{1}{\ell}\cdot\binom{\ell}{1}z^{\ell-1} + \frac{2}{\ell}\cdot\binom{\ell}{2}z^{\ell-2}(1-z) + \cdots + \frac{\ell}{\ell}\cdot\binom{\ell}{\ell}z^0(1-z)^{\ell-1}\\ &= \binom{\ell-1}{0}z^{\ell-1}(1-z)^0 + \binom{\ell-1}{1}z^{\ell-2}(1-z)^1 + \cdots + \binom{\ell-1}{\ell-1}z^0(1-z)^{\ell-1}\\ &= (z+1-z)^{\ell-1}\,. \end{aligned} \tag{4.5.34}$$

Indeed we use a degree-reduced Bernstein bases here, $\binom{\ell-1}{k}z^k(1-z)^{\ell-1-k}$, to express 1. In any case, the coefficients of 1 give us our $\boldsymbol{X}$ vector: $\boldsymbol{X}\boldsymbol{R}^{-1}\boldsymbol{Y} = \boldsymbol{P}^{-1}(z)$. □

Theorem 4.17 (Lagrange Basis)**.** *If $\boldsymbol{P}(z) \in \mathbb{C}^{n\times n}$ is of degree at most ℓ, and takes the values $\boldsymbol{\rho}_k \in \mathbb{C}^{n\times n}$ at the $\ell+1$ distinct nodes $z = \tau_k$, $0 \le k \le \ell$, i.e $\boldsymbol{P}(\tau_k) = \boldsymbol{\rho}_k \in \mathbb{C}^{n\times n}$, and the reciprocal of the node polynomial $w(z) = \prod_{k=0}^{\ell}(z-\tau_k)$ has partial fraction expansion*

$$\frac{1}{w(z)} = \sum_{k=0}^{\ell}\frac{\beta_k}{z-\tau_k} \tag{4.5.35}$$

then a linearization for $\boldsymbol{P}(z)$ is $z\boldsymbol{C}_1 - \boldsymbol{C}_0$ where $\boldsymbol{C}_1 = \operatorname{diag}(\boldsymbol{0}_n, \boldsymbol{I}_n, \boldsymbol{I}_n, \cdots, \boldsymbol{I}_n)$ with $\ell+2$ diagonal blocks, so $N = (\ell+2)r$, and

$$\boldsymbol{C}_0 = \left[\begin{array}{c|cccccc} 0 & -\boldsymbol{\rho}_0 & -\boldsymbol{\rho}_1 & -\boldsymbol{\rho}_2 & \cdots & -\boldsymbol{\rho}_\ell \\ \hline \beta_0\boldsymbol{I}_n & \tau_0\boldsymbol{I}_n & & & & \\ \beta_1\boldsymbol{I}_n & & \tau_1\boldsymbol{I}_n & & & \\ \beta_2\boldsymbol{I}_n & & & \tau_2\boldsymbol{I}_n & & \\ \vdots & & & & \ddots & \\ \beta_\ell\boldsymbol{I}_n & & & & & \tau_\ell\boldsymbol{I} \end{array}\right]. \tag{4.5.36}$$

Moreover, if $\boldsymbol{Y} = \begin{bmatrix}\boldsymbol{I}_n & 0 & 0 & \cdots & 0\end{bmatrix}^T$ and $\boldsymbol{X} = \begin{bmatrix}0_n & \boldsymbol{I}_n & \boldsymbol{I}_n & \cdots & \boldsymbol{I}_n\end{bmatrix}$ then $\boldsymbol{X}(z\boldsymbol{C}_1 - \boldsymbol{C}_0)^{-1}\boldsymbol{Y} = \boldsymbol{P}^{-1}(z)$ where $z \in \Lambda(\boldsymbol{P})$.

Proof. Again we use the Schur complement: $\boldsymbol{S} = \boldsymbol{A} - \boldsymbol{B}\boldsymbol{D}^{-1}\boldsymbol{C}$ where here

$$\boldsymbol{A} = \boldsymbol{0}_n \tag{4.5.37}$$

$$\boldsymbol{B} = -\begin{bmatrix}\boldsymbol{\rho}_0 & \boldsymbol{\rho}_1 & \cdots & \boldsymbol{\rho}_\ell\end{bmatrix} \tag{4.5.38}$$

$$\boldsymbol{D}^{-1} = \operatorname{diag}\left(\frac{1}{z-\tau_0}\boldsymbol{I}_n, \frac{1}{z-\tau_1}\boldsymbol{I}_n, \cdots, \frac{1}{z-\tau_\ell}\boldsymbol{I}_n\right) \tag{4.5.39}$$

$$\boldsymbol{C} = \begin{bmatrix}\beta_0\boldsymbol{I}_n \\ \beta_1\boldsymbol{I}_n \\ \vdots \\ \beta_\ell\boldsymbol{I}_n\end{bmatrix} \tag{4.5.40}$$

So

$$\boldsymbol{S} = \sum_{k=0}^{\ell}\frac{\beta_k}{z-\tau_k}\boldsymbol{\rho}_k = w(z)^{-1}\boldsymbol{P}(z) \tag{4.5.41}$$

from the first barycentric formula [4].
Note the first column of $\boldsymbol{R}^{-1}(z)$ is $\begin{bmatrix} \boldsymbol{S}^{-1} \\ -\boldsymbol{C}\boldsymbol{D}^{-1}\boldsymbol{S}^{-1} \end{bmatrix}$ or

$$\begin{bmatrix} w(z)\boldsymbol{P}^{-1}(z) \\ \left(\dfrac{\beta_0}{z-\tau_0}\right) w(z)\boldsymbol{P}^{-1}(z) \\ \left(\dfrac{\beta_1}{z-\tau_1}\right) w(z)\boldsymbol{P}^{-1}(z) \\ \vdots \\ \left(\dfrac{\beta_\ell}{z-\tau_\ell}\right) w(z)\boldsymbol{P}^{-1}(z) \end{bmatrix} \tag{4.5.42}$$

Note that $\sum_{k=0}^{\ell} \dfrac{\beta_k}{z-\tau_k} = \dfrac{1}{w(z)}$, so

$$\begin{bmatrix} 0 & \boldsymbol{I}_n & \boldsymbol{I}_n & \cdots & \boldsymbol{I}_n \end{bmatrix} \cdot \boldsymbol{R}^{-1} \begin{bmatrix} \boldsymbol{I}_n \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \left(\sum_{k=0}^{\ell} \frac{\beta_k}{z-\tau_k} \right) w(z)\boldsymbol{P}^{-1}(z) = \boldsymbol{P}^{-1}(z) \tag{4.5.43}$$

□

Theorem 4.18. *In the Hermite interpolational bases on $m+1$ nodes each with coefficiency s_i, so the degree ℓ is at most $\ell = -1 + \sum_{k=0}^{m} s_k$, the barycentric weights are*

$$\frac{1}{w(z)} = \sum_{i=0}^{m} \sum_{j=0}^{s_i-1} \frac{\beta_{ij}}{(z-\tau_i)^{j+1}} \tag{4.5.44}$$

As in the Lagrange case, $\boldsymbol{C}_1 = \operatorname{diag}(0, \boldsymbol{I}_n, \cdots, \boldsymbol{I}_n)$. $\boldsymbol{C}_0$ is as below:

$$\boldsymbol{C}_0 = \left[\begin{array}{c|cccc} 0 & -\hat{\boldsymbol{\rho}}_0 & -\hat{\boldsymbol{\rho}}_1 & \cdots & -\hat{\boldsymbol{\rho}}_m \\ \hline \beta_{0,s_0-1}\boldsymbol{I}_n & \boldsymbol{J}_0^T & & & \\ \beta_{0,s_0-2}\boldsymbol{I}_n & & \boldsymbol{J}_1^T & & \\ \vdots & & & \ddots & \\ \beta_{m,s_m-1} & & & & \boldsymbol{J}_m^T \end{array}\right] \tag{4.5.45}$$

where each block per node of data is collected in the $n \times m\ell$ block matrix

$$\hat{\boldsymbol{\rho}}_i = \begin{bmatrix} \boldsymbol{\rho}_{i,s_i-1} & \boldsymbol{\rho}_{i,s_i-2} & \cdots & \boldsymbol{\rho}_{i,0} \end{bmatrix} . \tag{4.5.46}$$

Each diagonal node block is a tensor product of a transposed Jordan block:

$$\boldsymbol{J}_i = \begin{bmatrix} \tau_i\boldsymbol{I}_n & & & & \\ \boldsymbol{I}_n & \tau_i\boldsymbol{I}_n & & & \\ & \boldsymbol{I}_n & \tau_i\boldsymbol{I}_n & & \\ & & \ddots & \ddots & \\ & & & \boldsymbol{I}_n & \tau_i\boldsymbol{I}_n \end{bmatrix} . \tag{4.5.47}$$

This form arises naturally on letting distinct Lagrange nodes flow together in a limit.

Express 1 *as a polynomial in this basis. Then* $1 \longleftrightarrow \rho_{00} = 1, \rho_{10} = 1, \cdots, \rho_{n0} = 1$ *and all other components are zero. Put*

$$\boldsymbol{X} = \begin{bmatrix} 0 & \underbrace{0 & \cdots & 0 & 1}_{s_0 \text{ entries}} & \underbrace{0 & \cdots & 0 & 1}_{s_1 \text{ entries}} & \cdots & 1 \end{bmatrix} \otimes \boldsymbol{I}_n \tag{4.5.48}$$

and $\boldsymbol{Y} = \begin{bmatrix} \boldsymbol{I}_n & 0 & 0 & \cdots & 0 \end{bmatrix}$.
A similar but more involved computation than in Theorem 4.17 gives

$$\boldsymbol{S} = \frac{1}{w(z)}\boldsymbol{P}(z) = \sum_{i=0}^{m}\sum_{j=0}^{s_i-1}\sum_{k=0}^{j}\beta_{ij}\boldsymbol{\rho}_{ik}(z-\tau_i)^{k-j-1} \tag{4.5.49}$$

and $\boldsymbol{D}^{-1}\boldsymbol{C}$ *contains just the correct powers of* $(z-\tau_i)$ *divided into* β_{ij} *to make the sums come out right; the inverse of the block*

$$\begin{bmatrix} (z-\tau_0)\boldsymbol{I}_n & & & & \\ -\boldsymbol{I}_n & (z-\tau_0)\boldsymbol{I}_n & & & \\ & -\boldsymbol{I}_n & \ddots & & \\ & & \ddots & & \\ & & & -\boldsymbol{I}_n & (z-\tau_0)\boldsymbol{I}_n \end{bmatrix} \tag{4.5.50}$$

is

$$\begin{bmatrix} \frac{1}{z-\tau_0}\boldsymbol{I}_n & & & \\ \frac{1}{(z-\tau_0)^2}\boldsymbol{I}_n & \frac{1}{z-\tau_0}\boldsymbol{I}_n & & \\ \frac{1}{(z-\tau_0)^3}\boldsymbol{I}_n & \frac{1}{(z-\tau_0)^2}\boldsymbol{I}_n & \frac{1}{z-\tau_0}\boldsymbol{I}_n & \\ \vdots & & \ddots & \\ \frac{1}{(z-\tau_0)^{s_0}}\boldsymbol{I}_n & & & \frac{1}{z-\tau_0}\boldsymbol{I}_n \end{bmatrix}. \tag{4.5.51}$$

and thus each block is reminiscent of Theorem 4.15, in fact.

Remark 4.19. In every case $\boldsymbol{X} = [\text{coefficients of } 1] \otimes \boldsymbol{I}$, $\boldsymbol{Y} = [1, 0, \cdots, 0] \otimes \boldsymbol{I}$. This is in agreement with our universal proof in Section 4.2.

4.6 Examples

In this section, we will show some experiments done in Maple 2017 to demonstrate that the standard triples introduced in Section 4.3 work for the different bases. We wrote our own code for constructing the linearizations rather than using Maple's built-in `CompanionMatrix` function since the result of the built-in function is the flipped and transposed version of the linearizations compared to the structure in this paper.

For the following examples, we check the correctness of the standard triple for each of the following examples by rearranging the resolvent form

$$\begin{aligned} \boldsymbol{P}^{-1}(z) &= \boldsymbol{X}\left(z\boldsymbol{C}_1 - \boldsymbol{C}_0\right)^{-1}\boldsymbol{Y} \\ \boldsymbol{I}_n &= \boldsymbol{X}\left(z\boldsymbol{C}_1 - \boldsymbol{C}_0\right)^{-1}\boldsymbol{Y}\boldsymbol{P}(z)\,. \end{aligned}$$

Since these computations are done exactly, the result will exactly equal the identity matrix. For the Lagrange basis example, since we construct our linearizations using τ and

ρ instead of the matrix polynomial itself, $\boldsymbol{P}(z)$ is constructed using the barycentric Lagrange interpolation formula, which can be derived from Equation (4.5.41). The Hermite interpolational basis examples are handled similarly to the Lagrange case, where $\boldsymbol{P}(z)$ is the Hermite interpolation polynomial, which can be derived from Equation (4.5.49).

4.6.1 Bases with three-term recurrence relations

Example 4.1 (Chebyshev basis of the first kind)**.**

$$\begin{aligned}\boldsymbol{P}(z) = &\begin{bmatrix} 1/5 & 7/100 \\ -93/200 & -29/200 \end{bmatrix} T_0(z) + \begin{bmatrix} 53/300 & 7/60 \\ 2/25 & 3/50 \end{bmatrix} T_1(z) \\ &+ \begin{bmatrix} -9/80 & -13/80 \\ 57/400 & -47/400 \end{bmatrix} T_2(z) + \begin{bmatrix} -3/250 & -31/500 \\ -77/500 & 27/250 \end{bmatrix} T_3(z)\,.\end{aligned} \tag{4.6.1}$$

The standard triple for Equation (4.6.1) *is*

$$\boldsymbol{C}_0 = \begin{bmatrix} -21/200 & -29/400 & -107/750 & -17/50 & -73/200 & 1/25 \\ -13/400 & -1/400 & -49/300 & -283/750 & -27/200 & 9/25 \\ 1/2 & 0 & 0 & 0 & 1/2 & 0 \\ 0 & 1/2 & 0 & 0 & 0 & 1/2 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

$$\boldsymbol{C}_1 = \begin{bmatrix} -33/250 & -9/25 & 0 & 0 & 0 & 0 \\ 3/25 & -37/250 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\boldsymbol{X} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad \boldsymbol{Y} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}.$$

Then,

$$\boldsymbol{X}\left(z\boldsymbol{C}_1 - \boldsymbol{C}_0\right)^{-1}\boldsymbol{Y}\boldsymbol{P}(z) = \boldsymbol{I}_2\,.$$

which indicates that the standard triple is correct.

Example 4.2 (Newton Interpolational Basis)**.**

$$\tau = \left[\operatorname{seq}\left(\cos\left(\frac{\pi\cdot k}{3}\right), k = 0..3\right)\right] = [1, 1/2, -1/2, -1] \tag{4.6.2}$$

$$\begin{aligned}\boldsymbol{P}(z) = &\begin{bmatrix} 6 & 25 \\ -1 & 5 \end{bmatrix} \prod_{j=0}^{0}(z-\tau_j) + \begin{bmatrix} -80/3 & 25/3 \\ 43/3 & 94/3 \end{bmatrix} \prod_{j=0}^{1}(z-\tau_j) \\ &+ \begin{bmatrix} 77/4 & 31/4 \\ 9/4 & -25/2 \end{bmatrix} \prod_{j=0}^{2}(z-\tau_j) + \begin{bmatrix} 86/5 & -61/5 \\ 4 & -48/5 \end{bmatrix} \prod_{j=0}^{3}(z-\tau_j)\end{aligned} \tag{4.6.3}$$

The standard triple for Equation (4.6.3) *is*

$$\boldsymbol{C}_0 = \begin{bmatrix} -557/20 & -33/20 & 80/3 & -25/3 & -6 & -25 \\ -17/4 & 173/10 & -43/3 & -94/3 & 1 & -5 \\ 1 & 0 & 1/2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1/2 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \end{bmatrix} \quad \boldsymbol{C}_1 = \begin{bmatrix} \frac{86}{5} & -\frac{61}{5} & 0 & 0 & 0 & 0 \\ 4 & -\frac{48}{5} & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\boldsymbol{X} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad \boldsymbol{Y} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}.$$

4.6.2 Bernstein Basis

Example 4.3 (Non-singular leading coefficient case)**.**

$$\begin{aligned} \boldsymbol{P}(z) = &\begin{bmatrix} 4/25 & 99/100 \\ 9/100 & 3/5 \end{bmatrix} B_0^3(z) + \begin{bmatrix} -17/25 & 11/50 \\ -67/100 & 7/50 \end{bmatrix} B_1^3(z) \\ &+ \begin{bmatrix} -59/100 & -31/50 \\ 3/25 & -33/100 \end{bmatrix} B_2^3(z) + \begin{bmatrix} 41/50 & 21/50 \\ 18/25 & 9/50 \end{bmatrix} B_3^3(z)\,. \end{aligned} \tag{4.6.4}$$

The standard triple for Equation (4.6.4) *is*

$$\boldsymbol{C}_0 = \begin{bmatrix} 59/100 & 31/50 & 17/25 & -11/50 & -4/25 & -99/100 \\ -3/25 & 33/100 & 67/100 & -7/50 & -9/100 & -3/5 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

$$\boldsymbol{C}_1 = \begin{bmatrix} 259/300 & 19/25 & 17/25 & -11/50 & -4/25 & -99/100 \\ 3/25 & 39/100 & 67/100 & -7/50 & -9/100 & -3/5 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 3 & 0 \\ 0 & 0 & 0 & 1 & 0 & 3 \end{bmatrix}$$

$$\boldsymbol{X} = \begin{bmatrix} 1/3 & 0 & 2/3 & 0 & 1 & 0 \\ 0 & 1/3 & 0 & 2/3 & 0 & 1 \end{bmatrix} \quad \boldsymbol{Y} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}.$$

Then,

$$\boldsymbol{X}\left(z\boldsymbol{C}_1 - \boldsymbol{C}_0\right)^{-1}\boldsymbol{Y}\boldsymbol{P}(z) = \boldsymbol{I}_2\,.$$

Example 4.4 (Singular leading coefficient case)**.**

$$\boldsymbol{P}(z) = \begin{bmatrix} 29/100 & -8/25 \\ 7/10 & -1/100 \end{bmatrix} B_0^3(z) + \begin{bmatrix} -41/50 & 41/100 \\ -7/10 & 91/100 \end{bmatrix} B_1^3(z) \\ + \begin{bmatrix} 9/10 & 19/100 \\ 4/5 & 22/25 \end{bmatrix} B_2^3(z) + \begin{bmatrix} 1 & 1 \\ 9851/1980 & 0 \end{bmatrix} B_3^3(z)\,. \tag{4.6.5}$$

Expressing Equation (4.6.5) *into the monomial basis, we have*

$$\boldsymbol{P}(z) = \begin{bmatrix} 29/100 & -8/25 \\ 7/10 & -1/100 \end{bmatrix} + \begin{bmatrix} 29/100 & -8/25 \\ 7/10 & -1/100 \end{bmatrix} z + \begin{bmatrix} 849/100 & -57/20 \\ 87/10 & -57/20 \end{bmatrix} z^2 + \begin{bmatrix} -89/20 & 99/50 \\ -89/396 & 1/10 \end{bmatrix} z^3\,.$$

Taking the determinant of the leading coefficient

$$\det\left(\begin{bmatrix} -89/20 & 99/50 \\ -89/396 & 1/10 \end{bmatrix}\right) = (-89/20)\,(1/10) - (99/50)\,(-89/396) = 0\,,$$

we can observe that leading coefficient is singular, and thus, this matrix polynomial is non-monic. The standard triple for Equation (4.6.5) *is*

$$\boldsymbol{C}_0 = \begin{bmatrix} -9/10 & -19/100 & 41/50 & -41/100 & -29/100 & 8/25 \\ -4/5 & -22/25 & 7/10 & -91/100 & -7/10 & 1/100 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

$$\boldsymbol{C}_1 = \begin{bmatrix} -17/30 & 43/300 & 41/50 & -41/100 & -29/100 & 8/25 \\ 5099/5940 & -22/25 & 7/10 & -91/100 & -7/10 & 1/100 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 3 & 0 \\ 0 & 0 & 0 & 1 & 0 & 3 \end{bmatrix}$$

$$\boldsymbol{X} = \begin{bmatrix} 1/3 & 0 & 2/3 & 0 & 1 & 0 \\ 0 & 1/3 & 0 & 2/3 & 0 & 1 \end{bmatrix} \quad \boldsymbol{Y} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}.$$

Then,

$$\boldsymbol{X}\left(z\boldsymbol{C}_1 - \boldsymbol{C}_0\right)^{-1}\boldsymbol{Y}\boldsymbol{P}(z) = \boldsymbol{I}_2\,.$$

4.6.3 Lagrange Basis

Example 4.5.

$$\tau = \left[\operatorname{seq}\left(\cos\left(\frac{\pi\cdot k}{2}\right), k = 0..2\right)\right] = [1, 0, -1] \tag{4.6.6}$$
$$\rho = [\boldsymbol{I}_2, \boldsymbol{I}_2, \boldsymbol{I}_2] = \left[\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\right]$$

$$\boldsymbol{C}_0 = \begin{bmatrix} 0 & 0 & -1 & 0 & -1 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 & 0 & -1 & 0 & -1 \\ 1/2 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1/2 & 0 & 1 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 1/2 & 0 & 0 & 0 & 0 & 0 & -1 \end{bmatrix} \quad \boldsymbol{C}_1 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\boldsymbol{X} = \begin{bmatrix} 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix} \quad \boldsymbol{Y} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}.$$

Using the barycentric Lagrange interpolation formula, we construct our matrix polynomial

$$\begin{aligned} \boldsymbol{P}(z) &= \begin{bmatrix} (z-1)\,z\,(z+1)\left(\frac{1}{2(z-1)} - \frac{1}{z} + \frac{1}{2(z+1)}\right) & 0 \\ 0 & (z-1)\,z\,(z+1)\left(\frac{1}{2(z-1)} - \frac{1}{z} + \frac{1}{2(z+1)}\right) \end{bmatrix} \\ &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \end{aligned}$$

that corresponds to the given τ and ρ from Equation 4.6.6. Therefore, $\boldsymbol{P}^{-1}(z)$

$$\boldsymbol{X}\,(z\boldsymbol{C}_1 - \boldsymbol{C}_0)^{-1}\,\boldsymbol{Y} = \boldsymbol{I}_2$$

4.6.4 Hermite Interpolational Basis

Example 4.6 (Polynomial case)**.** *Let*

$$\tau = \left[-1, -\frac{1}{2}, \frac{1}{2}, 1\right] \tag{4.6.7}$$

and

z	$P(z)$	$P'(z)$	$P''(z)$
$\tau_0 = -1$	1	0	0
$\tau_1 = -\frac{1}{2}$	1		
$\tau_2 = \frac{1}{2}$	1		
$\tau_3 = 1$	1	0	

Note that this polynomial is identically 1*: its values at all nodes are* 1*, and all derivatives at all nodes are* 0*. This demonstrates explicitly that the degree of the polynomial is not*

necessarily revealed by the grade, which here is $\ell = 5$. The standard triple is then

$$\boldsymbol{C}_0 = \begin{bmatrix} 0 & 0 & -1 & -1 & -1 & 0 & 0 & -1 \\ 1/6 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -25/36 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 32/27 & 0 & 0 & 1/2 & 0 & 0 & 0 & 0 \\ -32/9 & 0 & 0 & 0 & -1/2 & 0 & 0 & 0 \\ 1/3 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 11/9 & 0 & 0 & 0 & 0 & 1 & -1 & 0 \\ 331/108 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \end{bmatrix} \quad \boldsymbol{C}_1 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\boldsymbol{X} = \begin{bmatrix} 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix} \quad \boldsymbol{Y} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

Using the first barycentric representation, the Hermite interpolation polynomial of the given data is

$$\begin{aligned} P(z) = {} & (z+1)^3 \left(z + \frac{1}{2}\right)\left(z - \frac{1}{2}\right)(z-1)^2 \cdot \\ & \left(\frac{331}{108z+108} + \frac{11}{9\,(z+1)^2} + \frac{1}{3}\,(z+1)^{-3} - \frac{32}{9z+\frac{9}{2}} + \frac{32}{27z-\frac{27}{2}} - \frac{25}{36z-36} + \frac{1}{6}\,(z-1)^{-2}\right) \\ & = 1\,, \end{aligned}$$

as discussed. Therefore the linearization has no finite eigenvalues, in exact arithmetic. Numerically, it can be expected to have eigenvalues around $O(1/\mu)^{1/7}$ where μ is the unit roundoff; here the exponent is 7, two more than the grade even though two of the spurious eigenvalues at infinity are detected and removed precisely [25]. Indeed that is what occurs (calculations not shown here). Returning to the example, calculating the resolvent form gives

$$\boldsymbol{X}\,(z\boldsymbol{C}_1 - \boldsymbol{C}_0)^{-1}\,\boldsymbol{Y} = 1\,,$$

and therefore (due to multiplying this by $P(z) = 1$), this shows that the standard triple for the Hermite interpolating basis is correct.

Example 4.7 (Matrix polynomial case)**.** *Let*

$$\tau = [0, 1]$$

and

z	$\boldsymbol{P}(z)$	$\boldsymbol{P}'(z)$
$\tau_0 = 0$	$\begin{bmatrix} -1 & 0 \\ -1 & 1 \end{bmatrix}$	
$\tau_1 = 1$	$\begin{bmatrix} 0 & 1 \\ 1 & -1 \end{bmatrix}$	$\begin{bmatrix} 1 & -1 \\ -1 & 0 \end{bmatrix}$

Then, the standard triple is

$$\boldsymbol{C}_0 = \begin{bmatrix} 0 & 0 & -1 & 1 & 0 & -1 & 1 & 0 \\ 0 & 0 & 1 & 0 & -1 & 1 & 1 & -1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \boldsymbol{C}_1 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\boldsymbol{X} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \end{bmatrix} \quad \boldsymbol{Y} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}.$$

The Hermite interpolating polynomial is

$$\boldsymbol{P}(z) = \begin{bmatrix} z-1 & -2z^2+3z \\ -3z^2+5z-1 & 2z^2-4z+1 \end{bmatrix}$$

and the resolvent form is

$$\boldsymbol{X}\left(z\boldsymbol{C}_1 - \boldsymbol{C}_0\right)^{-1}\boldsymbol{Y} = \begin{bmatrix} \dfrac{-2z^2+4z-1}{6z^4-21z^3+23z^2-8z+1} & \dfrac{-2z^2+3z}{6z^4-21z^3+23z^2-8z+1} \\ \dfrac{-3z^2+5z-1}{6z^4-21z^3+23z^2-8z+1} & \dfrac{-z+1}{6z^4-21z^3+23z^2-8z+1} \end{bmatrix}.$$

Then,

$$\boldsymbol{X}\left(z\boldsymbol{C}_1 - \boldsymbol{C}_0\right)^{-1}\boldsymbol{Y}\boldsymbol{P}(z) = \boldsymbol{I}_2\,,$$

which indicates that the standard triples is correct.

4.7 Concluding remarks

The generalized standard triple (or standard quadruple, if you prefer) that we propose in this paper for convenience in algebraic linearization may have other uses. As pointed out on p. 28 of [19] many of the properties stated in that work for monic polynomials are valid for non-monic polynomials with the appropriate changes made. Some caution with the results of this paper are thus mandated.

We have here *defined* these generalized standard triples simply by the resolvent representation for the matrix polynomial Equation (4.1.10), and only for linearizations, which is all we need for algebraic linearization.

The main theorem of the paper, namely Theorem 4.8, gives a universal way to construct this generalized standard triple in any polynomial basis. We also gave explicit instructions for this construction using any of several polynomial bases, for convenience, together with separate proofs using the Schur complement, which may give insight for further work in this area.

We have also recorded a number of smaller results. In Section 4.3.2.1 we give a new reversal for the Bernstein linearization, one which may have slightly superior numerical qualities. We there showed strict equivalence of the appropriate matrices, giving a new proof that the Bernstein linearization is a strong one. In Section 4.1.3 we have sketched an explicit construction for matrices $\boldsymbol{E}$ and $\boldsymbol{F}$ showing that *algebraic* linearizations are, in fact, linearizations, with $\boldsymbol{E}(z\boldsymbol{D}_H - \boldsymbol{H})\boldsymbol{F} = \operatorname{diag}(\boldsymbol{P}(z), \boldsymbol{I}, \ldots, \boldsymbol{I})$. We have also given new constructions for matrices $\boldsymbol{E}$ and $\boldsymbol{F}$ which likewise show that the companions for the Lagrange and Hermite interpolational bases are, in fact, linearizations (of matrix polynomials of higher grade). A proof for Lagrange interpolational bases was given already in [2], where indeed the linearization was proved to be strong, but the result for Hermite interpolational bases is new to this paper. We also used Hermite Form computations to give a new (to us) pair $\boldsymbol{E}$ and $\boldsymbol{F}$ for the ordinary monomial basis.

Acknowledgments

We acknowledge the support of Western University, The National Science and Engineering Research Council of Canada, the Ontario Graduate Scholarship (OGS) program, the University of Alcalá, the Ontario Research Centre of Computer Algebra, and the Rotman Institute of Philosophy. Part of this work was developed while RMC was visiting the University of Alcalá, in the frame of the project Giner de los Rios. The authors would also like to thank Peter Lancaster for teaching RMC long ago the value of the 5×5 example. Similarly we thank John C. Butcher for the proper usage of the word "interpolational". We thank Françoise Tisseur for her thorough comments on an earlier version of this paper. Finally, we thank an anonymous referee for the idea of the proof of Theorem 4.8.

Bibliography

[1] Al-Ammari, Maha and Tisseur, Francoise. Standard triples of structured matrix polynomials. *Linear algebra and its applications, Elsevier*, 437(3):817–834, 2012

[2] Amir Amiraslani, Robert M. Corless, and Peter Lancaster. Linearization of matrix polynomials expressed in polynomial bases. *IMA Journal of Numerical Analysis*, 29(1):141–157, 2008.

[3] Stephen Barnett. Some applications of the comrade matrix. *International Journal of Control*, 21(5):849–855, 1975.

[4] Jean-Paul Berrut and Lloyd N. Trefethen. Barycentric Lagrange interpolation. *SIAM Review*, 46(3):501–517, 2004.

[5] Timo Betcke, Nicholas J. Higham, Volker Mehrmann, Christian Schröder, and Françoise Tisseur. NLEVP: A collection of nonlinear eigenvalue problems. *ACM Transactions on Mathematical Software (TOMS)*, 39(2):7, 2013.

[6] J. M. Carnicer, Y. Khiar, and J. M. Peña. Optimal stability of the Lagrange formula and conditioning of the Newton formula. *Journal of Approximation Theory*, 2017.

[7] Eunice Y. S. Chan, Robert M. Corless, Laureano Gonzalez-Vega, J. Rafael Sendra, and Juana Sendra. Algebraic linearizations for matrix polynomials. *Linear Algebra and its Applications*, 563:373–399, 2019.

[8] Robert M. Corless and Nicolas Fillion. Polynomial and rational interpolation. In *A Graduate Introduction to Numerical Methods*, pages 331–401. Springer, 2013.

[9] Robert M Corless, Nargol Rezvani, and Amirhossein Amiraslani. Pseudospectra of matrix polynomials that are expressed in alternative bases. *Mathematics in Computer Science*, 1(2):353–374, 2007.

[10] Robert M. Corless and Stephen M. Watt. Bernstein bases are optimal, but, sometimes, Lagrange bases are better. In *Proceedings of SYNASC, Timisoara*, pages 141–153. MIRTON Press, 2004.

[11] Froilán M Dopico, Piers W. Lawrence, Javier Pérez, and Paul Van Dooren. Block Kronecker linearizations of matrix polynomials and their backward errors. *Numerische Mathematik*, 140(2):373–426, 2018.

[12] Rida T. Farouki. The Bernstein polynomial basis: A centennial retrospective. *Computer Aided Geometric Design*, 29(6):379–419, 2012.

[13] Rida T. Farouki and T. Goodman. On the optimal stability of the Bernstein basis. *Mathematics of Computation of the American Mathematical Society*, 65(216):1553–1566, 1996.

[14] Rida T. Farouki and V. T. Rajan. On the numerical condition of polynomials in Bernstein form. *Computer Aided Geometric Design*, 4(3):191–216, 1987.

[15] Heike Faßbender and Philip Saltenberger. On vector spaces of linearizations for matrix polynomials in orthogonal bases. *Linear Algebra and its Applications*, 525:59–83, 2017.

[16] Walter Gautschi. *Orthogonal polynomials in MATLAB: Exercises and Solutions*, volume 26. SIAM, 2016.

[17] Israel Gohberg, Peter Lancaster, and Leiba Rodman. *Matrix polynomials*. Academic Press, New York, 1982.

[18] Israel Gohberg, Peter Lancaster, and Leiba Rodman. *Indefinite Linear Algebra and Applications*. Springer, 2005.

[19] Israel Gohberg, Peter Lancaster, and Leiba Rodman. *Matrix Polynomials*. SIAM Classics in Applied Mathematics, 2009.

[20] Stefan Güttel and Françoise Tisseur. The nonlinear eigenvalue problem. *Acta Numerica*, 26:1–94, 2017.

[21] David J. Jeffrey and Robert M. Corless. Linear algebra in maple. In Leslie Hogben, editor, *Handbook of Linear Algebra*, chapter 89. Chapman and Hall/CRC, 2013.

[22] Guðbjörn F. Jónsson. *Eigenvalue methods for accurate solution of polynomial equations.* PhD thesis, Center for Applied Mathematics, Cornell University, Ithaca, NY, 2001.

[23] Guðbjörn F. Jónsson and Stephen Vavasis. Solving polynomials with small leading coefficients. *SIAM Journal on Matrix Analysis and Applications*, 26(2):400–414, 2004.

[24] V. N. Kublanovskaya. Methods and algorithms of solving spectral problems for polynomial and rational matrices. *Journal of Mathematical Sciences*, 96(3):3085–3287, 1999.

[25] Piers W. Lawrence and Robert M. Corless. Numerical stability of barycentric Hermite root-finding. In *Proceedings of the 2011 International Workshop on Symbolic-Numeric Computation*, pages 147–148. ACM, 2012.

[26] Jörg Liesen and Christian Mehl. Matrix polynomials. In Leslie Hogben, editor, *Handbook of Linear Algebra*, chapter 18. Chapman and Hall/CRC, 2013.

[27] D. Steven Mackey, Niloufer Mackey, Christian Mehl, and Volker Mehrmann. Vector spaces of linearizations for matrix polynomials. *SIAM Journal on Matrix Analysis and Applications*, 28(4):971–1004, 2006.

[28] D. Steven Mackey, Niloufer Mackey, and Françoise Tisseur. Polynomial eigenvalue problems: Theory, computation, and structure. In *Numerical Algebra, Matrix Theory, Differential-Algebraic Equations and Control Theory*, pages 319–348. Springer, 2015.

[29] D. Steven Mackey and Vasilije Perović. Linearizations of matrix polynomials in Bernstein bases. *Linear Algebra and its Applications*, 501:162–197, 2016.

[30] Yuji Nakatsukasa, Vanni Noferini, and Alex Townsend. Vector spaces of linearizations for matrix polynomials: a bivariate polynomial approach. *SIAM Journal on Matrix Analysis and Applications*, 38(1):1–29, 2017.

[31] Leonardo Robol, Raf Vandebril, and Paul Van Dooren. A framework for structured linearizations of matrix polynomials in various bases. *SIAM Journal on Matrix Analysis and Applications*, 38(1):188–216, 2017.

[32] Arne Storjohann. Computation of Hermite and Smith normal forms of matrices. *ETH Zurich*, 1994.

[33] Arne Storjohann. An $O(n^3)$ algorithm for the Frobenius normal form. In *Proceedings of the 1998 international symposium on Symbolic and algebraic computation*, pages 101–105. ACM, 1998.

[34] Olga Taussky and Hans Zassenhaus. On the similarity transformation between a matrix and its transpose. *Pacific Journal of Mathematics*, 9(3):893–896, 1959.

[35] Roel Van Beeumen, Wim Michiels, and Karl Meerbergen. Linearization of Lagrange and Hermite interpolating matrix polynomials. *IMA Journal of Numerical Analysis*, 35(2):909–930, 2015.

Chapter 5

On Parametric Linear System Solving

5.1 Introduction

Speaking in simple generalities, we say that symbolic computation is concerned with mathematical equations that contain *symbols*; symbols are used both for *variables*, which are typically to be solved for, and *parameters*, which are typically carried through and appear in the *solutions*, which are then interpreted as formulae: that is, objects that can be further studied, perhaps by varying the parameters. One prominent early researcher said that the difference between symbolic and numeric computation was merely a matter of *when* numerical values were inserted into the parameters: before the computation meant you were going to do things numerically, and after the computation meant you had done symbolic computation. The words "parameters" and "variables" are therefore not precisely descriptive, and can often be used interchangeably. Indeed as a matter of practice, polynomial equations can often be taken to have one subset of its symbols taken as variables rather than any other subset in quite strategic fashion: it may be better to solve for x as a function of y than to solve for y as a function of x.

In this paper we are concerned with systems of equations containing several symbols, some of which we take to be variables, and all the rest as parameters. More, we restrict our attention to problems in which the *variables* appear only linearly. Parameters are allowed to appear polynomially, of whatever degree.

Parametric linear systems (PLS) arise in many contexts, for instance in the analysis of the stability of equilibria in dynamical systems models such as occur in mathematical biology and other areas. Understanding the different potential kinds of dynamical behavior can be important for model selection as well as analysis. Another important area of interest is the role of parametric linear systems in dealing with the stability of the equilibria of parametric autonomous system of ordinary differential equations (see [25] and [11]). One particularly famous example is the Lotka-Volterra system which arises naturally from predator-prey equations. See also [24] and [23]. Other examples of the use of parametric linear system from science and engineering includes their application in computing the characteristic solutions for differential equations [8], dealing with colored Petri nets [13] and in operations research and engineering [9], [17], [21], [31]. Some problems in robotics [2] and certain modelling problems in mathematical biology, see e.g. [29], also can benefit from the ability to effectively solve PLS.

After some discussion of prior comprehensive solving work in Section 2, we proceed with formal problem and solution definitions for parametric linear systems (PLS) in section 3. Our primary tool for solving these is by way of *comprehensive triangular*

Smith normal form (CTSNF), which is introduced in section 4. The following section reduces PLS to CTSNF and section 6 describes the solution of CTSNF problems for the case of up to three parameters.

An application that seems at first to be of only theoretical interest is the computation of the *matrix logarithm*, or indeed any of several other matrix functions such as matrix square root. We briefly discuss this example in more detail with a pair of small matrices in Section 5.7.2. We also give other examples in section 7.

5.2 Prior Work

Interest in computation of the solution of PLS dates back to the beginning of symbolic computation. For instance, one of the first things users have requested of computer algebra systems is the explicit form of the inverse of a matrix containing only symbolic entries[1]: the user is then typically quite dissatisfied at the complexity of the answer if the dimension is greater than, say, 3. Of course, the determinant itself, which must appear in such an answer, has a factorial number of terms in it, and thus growth in the size of the answer must be more than exponential. Therefore the complexity of any algorithm to solve PLS must be at least exponential in the number of parameters.

An interesting pair of papers addressing the case of only one parameter is [1] and [15]. These papers assume full rank of the linear system—and thus compute the "generic" case when in fact there are isolated values of the parameter for which the rank drops—and use rational interpolation of the numerical solutions of specialized linear systems to recover this generic solution.

Many authors have sought comprehensive solutions—by which is meant complete coverage of all parametric regimes—through various means. One of the first serious methods was the matrix-minor based approach of William Sit [25], which enables practical solution of many problems of interest. Recently, the problem of computing the Jordan form of a parametric matrix once the Frobenius form is known has been attacked by using Regular Chains [4] and this has been moderately successful in practice. Simple methods and heuristics for linear systems containing parameters continue to generate interest, even when Regular Chains are used, such as in [3].

Other authors such as [30], [18], [20], [16], and [19] have tackled the even more difficult problem of computing the comprehensive solution of systems of *polynomial* equations containing parameters, and of course their methods can be applied to the linear equations being considered here.

By restricting our attention in this paper to linear problems and to those of three parameters or fewer we are able to guarantee better worst case performance (polynomially many solution regimes) and hope to provide better efficiency in many instances than is possible using those general-purpose approaches.

5.3 Definitions and Notation

Let F be a field and $Y = (y_1, \ldots, y_s)$ a list of parameters. Then $F[Y]$ is the ring of polynomials and $F(Y)$ is the field of rational functions in Y. For each tuple $a = (a_1, \ldots, a_s)$ in F^s, evaluation at a is a mapping $F[Y] \to F$. We will extend this mapping componentwise to polynomials, vectors, matrices, and sets thereof over $F[Y]$. We will use the Householder convention, typesetting matrices in upper case bold, e.g. $\boldsymbol{A}$, and lower case bold for vectors, e.g. $\vec{b}$.

[1] This is merely an anecdote, but one of the present authors attests that this really has happened.

For the most part, for such objects over $F[Y]$, we know Y from context and write $\boldsymbol{A}$ rather than $\boldsymbol{A}(Y)$, but write $\boldsymbol{A}(a)$ for the evaluation at $Y = a$.

For a set of polynomials, S, we will denote by $V(S)$ the variety of the ideal generated by S. This is the set of tuples a such that $f(a) = \{0\}$, for all $f \in S$. We will be concerned with pairs N, Z of polynomial sets, $N, Z \subset F[Y]$, defining a semialgebraic set in F^s consisting of those tuples a that evaluate to nonzero on N and to zero on Z. By a slight abuse of notation, we call this semialgebraic set $V(N, Z) = V(Z) \setminus V(N)$. Our inputs are polynomial in the parameters but the output coefficients in general are rational functions. The evaluation mapping extends partially to $F(Y)$: For a rational function $n(Y)/d(Y)$ in lowest terms (n and d relatively prime), the image $n(a)/d(a)$ is well defined so long as $d(a) \neq 0$.

Definition 5.1. The data for a **parametric linear system (PLS) problem** is matrix $\boldsymbol{A}$ and right hand side vector $\vec{b}$ over $F[Y]$, together with a semialgebraic constraint, $V(N, Z)$, with $N, Z \subset F[Y]$. Only of interest are those parameter value tuples in $V(N, Z)$, i.e., on which the polynomials in N are nonzero and the polynomials in Z are zero.

For the PLS problem $(\boldsymbol{A}, \vec{b}, N, Z)$, a **solution regime** is a tuple $(\vec{u}, \boldsymbol{B}, N', Z')$, with coefficients of $\vec{u}$ and $\boldsymbol{B}$ in $F(Y)$, such that, for all $a \in V(N', Z')$, $\vec{u}(a)$ is a solution vector and $\boldsymbol{B}(a)$ is a matrix whose columns form a nullspace basis for $\boldsymbol{A}(a)$.

A **PLS solution** is a set of solution regimes that covers $V(N, Z)$, which means, for PLS solution $\{(\vec{u}_i, \boldsymbol{B}_i, N_i, Z_i) | i \in 1, \ldots, k\}$, every parameter value assignment that satisfies the problem semialgebraic constraint N, Z also satisfies at least one regime semialgebraic constraint N_i, Z_i. In other words $V(N, Z) \subset \cup_{i=1}^{k} V(N_i, Z_i)$.

We call entries that *must* occur in any Z in the solution an *intrinsic* restriction, or singularity. We call the differing sets $V(N_i, Z_i)$ that may occur in covers of $V(N, Z)$ the *ramifications* of the cover.

We next give an example that illustrates the PLS definition and also sketches the prior approach to PLS given by William Sit in [25]. If, for $\boldsymbol{M}$ of size $r \times r$, $\boldsymbol{A}$ is $\begin{bmatrix} \boldsymbol{M} & \boldsymbol{B} \\ \boldsymbol{C} & \boldsymbol{D} \end{bmatrix}$, and conformally $\vec{b} = \begin{bmatrix} \vec{c} & \vec{d} \end{bmatrix}^T$, then a solution $\boldsymbol{u} = \begin{bmatrix} \vec{v} & \vec{w} \end{bmatrix}^T$ satisfies

$$\boldsymbol{M}\vec{v} + \boldsymbol{B}\vec{w} = \vec{c} \tag{5.3.1}$$

and

$$\boldsymbol{C}\vec{v} + \boldsymbol{D}\vec{w} = \vec{d}\,. \tag{5.3.2}$$

Under the condition that $\det(\boldsymbol{M})$ is nonzero and all larger minors of $\boldsymbol{A}$ are zero, equation (5.3.1) can be solved with specific solution $\vec{w} = 0$ and $\vec{v} = \boldsymbol{M}^{-1}\vec{c}$. Provided the system is consistent (equation (5.3.2) holds), we have the regime

$$(\begin{bmatrix} \vec{v} & \vec{w} \end{bmatrix}^T, \begin{bmatrix} -\boldsymbol{M}^{-1}\boldsymbol{B} \\ \boldsymbol{I} \end{bmatrix}, N, Z),$$

where $N = \{\det(\boldsymbol{M})\}$ and $Z = \{\text{all } (i+1)\times(i+1) \text{ minors of } \boldsymbol{A}\}$). Call solution regimes of this type *minor defined* regimes.

Since an $n \times n$ matrix has $\sum_{k=0}^{n} \binom{n}{k}^2 = \binom{2n}{n}$ minors, there are exponentially many minor defined regimes. However, some of these regimes may not be solutions due to inconsistency or it may be possible to combine several regimes into one. For instance if $\det(\boldsymbol{M})$ is a constant, and $\vec{b} = 0$, then all rank r solutions are covered by this one regime. Sit [25] has made a thorough study of minor defined regimes and their simplifications.

Another approach is to base solution regimes on the pivot choices in an LU decomposition. The simplest thing to do is to leave it to the user, although one has to also

inform the user through a proviso when this might be necessary [6]. That is, provide the generic answer, but also provide a description of the set N. A more sophisticated approach is developed in [3, 4] using the theory of regular chains and its implementation in Maple [18] to manage the algebraic conditions. For example a given matrix entry may be used as a pivot, with validity dependent on adding the polynomial to the non-zero part, N, of the semialgebraic set. For a comprehensive solution the case that that entry is zero must also be pursued. In the worst case, this leads to a tree of zero/nonzero choices of depth n and branching factor n.

5.4 Triangular Smith forms and degree bounds

In this paper we take a different approach, with the solution regimes arising from Hermite normal forms, of which triangular Smith forms are a special case. We give a system of solution regimes of polynomial size in the matrix dimension, n, and polynomial degree, d. Each regime is computed in polynomial time and the regime count is exponential only in the number of parameters. To use Hermite forms we will need to work over a principal ideal domain such as, for parameters x, y, $F(y)[x]$. We will restrict our input matrix to be polynomial in the parameters. This first lemma shows it is not a severe constraint.

Lemma 5.2. *Let $(\boldsymbol{A}, \vec{b}, N, Z)$ be a well defined PLS over field $F(Y)$, for parameter set Y, with $\boldsymbol{A} \in F(Y)^{m\times n}$ and $\vec{b} \in F(Y)^m$ with numerator and denominator degrees bounded by d in each parameter of Y. Well defined means that denominators of $\boldsymbol{A}, \vec{B}$ are in N. The problem is equivalent (same solutions) to one in which the entries of the matrix and vector are polynomial in the parameters Y, the dimension is the same, and the degrees are bounded by nd.*

Proof. Because the PLS is well defined, it is specified by N that all denominator factors of $\boldsymbol{A}(a), \vec{b}(a)$ are nonzero for $a \in V(N, Z)$. Let $\boldsymbol{L}$ be a diagonal matrix with the i-th diagonal entry being the least common multiple (lcm) of the denominators in row i of $\boldsymbol{A}$, $\vec{b}$. These lcms also evaluate to nonzero on $V(N, Z)$. It follows that $L(a)\boldsymbol{A}(a)\vec{u}(a) = \boldsymbol{L}(a)\vec{b}(a)$ if and only if $\boldsymbol{A}(a)\vec{u}(a) = \vec{b}(a)$. Thus the PLS $(\boldsymbol{LA}, \boldsymbol{L}\vec{b}, V(N, Z))$ is equivalent and its matrix and vector have polynomial entries of degrees bounded by nd. □

We will reduce PLS to triangular Smith normal form computations. The rest of this section concerns computation of triangular Smith normal form and bounds for the degrees of the form and its unimodular cofactor.

Definition 5.3. Given field K and variable x, a matrix $\boldsymbol{H}$ over $K[x]$ is in (reduced) **Hermite normal form** if it is upper triangular, its diagonal entries are monic, and, for each column in which the diagonal entry is nonzero, the off-diagonal entries are of lower degree than the diagonal entry. If each diagonal entry of $\boldsymbol{H}$ exactly divides all those below and to the right, then $\boldsymbol{H}$ is column equivalent to a diagonal matrix with the same diagonal entries (its Smith normal form). An equivalent condition is that, for each i, the greatest common divisor of the $i \times i$ minors in the leading i columns equals the greatest common divisor of all $i \times i$ minors. Following Storjohann[27, Section 8,Definition 8.2] we call such a Hermite normal form a **triangular Smith normal form**. It will be the central tool in our PLS solution.

For notational simplicity, we've left out the possibility of echelon structure in a Hermite normal form. We will talk of Hermite normal forms only for matrices having leading columns independent up to the rank of the matrix. Every such matrix over $K[x]$

is row equivalent to a unique matrix in Hermite form as defined above. For given $\boldsymbol{A}$ we have $\boldsymbol{UA} = \boldsymbol{H}$, with $\boldsymbol{U}$ unimodular, i.e. $\det(\boldsymbol{U}) \in K^*$, and $\boldsymbol{H}$ in Hermite form. If $\boldsymbol{A}$ is nonsingular, the unimodular cofactor $\boldsymbol{U}$ is unique and has determinant $1/c$, where c is the leading coefficient of $\det(\boldsymbol{A})$. This follows since $\det(\boldsymbol{U})\det(\boldsymbol{A}) = \det(\boldsymbol{H})$, which is monic.

The next definition and lemma concern assurance that Hermite form computation will yield a triangular Smith form.

Definition 5.4. Call a matrix **nice** if its Hermite form is a triangular Smith form (each diagonal entry exactly divides those below and to the right). In particular, a nice matrix has leading columns independent up to the rank.

There is always a column transform (unimodular matrix $\boldsymbol{R}$ applied from the right) such that $\boldsymbol{AR}$ is nice. The following fact, proven in [14] shows that a random transform over F suffices with high probability.

Fact 1. *Let $\boldsymbol{A}$ be a $m \times n$ matrix over $K[x]$ of degree in x at most d. Let $\boldsymbol{R}$ be a unit lower triangular matrix with below diagonal elements chosen from subset S of K uniformly at random. Then $\boldsymbol{AR}$ is nice over $K[x]$ with probability at least $1 - 4n^3d/|S|$.*

Note that $\deg_x(\boldsymbol{AR}) = \deg_x(\boldsymbol{A})$ and, for $K = F(y), \boldsymbol{A} \in F[y,x]^{m\times n}$ and $S \subset F$ we also have $\deg_y(\boldsymbol{AR}) = \deg_y(\boldsymbol{A})$.

We continue with analysis of degree bounds for Hermite forms of matrices, particularly degree bounds for triangular Smith forms of nice matrices. The first result needed is the following fact from [10]. Through the remainder of this paper we will employ "soft O" notation, where, for functions $f, g \in \mathbb{R}^k \to \mathbb{R}$ we write $f = O\tilde{}(g)$ if and only if $f = O(g \cdot \log^c |g|)$ for some constant $c > 0$.

Fact 2. *Let F be a field, x, y parameters, and let $\boldsymbol{A}$ be in $F[y,x]^{n\times n}$, nonsingular, with $\deg_x(\boldsymbol{A}) \le d$, $\deg_y(\boldsymbol{A}) \le e$. Over $F(y)[x]$, let $\boldsymbol{H}$ the unique Hermite form row equivalent to $\boldsymbol{A}$ and $\boldsymbol{U}$ be the unique unimodular cofactor such that $\boldsymbol{UA} = \boldsymbol{H}$. The coefficients of the entries of $\boldsymbol{H}$, $\boldsymbol{U}$ are rational functions of y. Let Δ be the least common multiple of the denominators of the coefficients in $\boldsymbol{H}$, $\boldsymbol{U}$, as expressed in lowest terms.*

(a) $\deg_x(\boldsymbol{U}) \le (n-1)d$ *and* $\deg_x(\boldsymbol{H}) \le nd$.

(b) $\deg_y(\mathrm{num}(\boldsymbol{H})), \deg_y(\mathrm{num}(\boldsymbol{U})) \le n^2de$ *(bounds both numerator and denominator degrees).*

(c) $\deg_y(\Delta) \le n^2de$.

(d) $\boldsymbol{H}$ and $\boldsymbol{U}$ can be computed in polynomial time: deterministically in $O\tilde{}(n^9d^4e)$ time and Las Vegas probabilistically (never returns incorrect result) in $O\tilde{}(n^7d^3e)$ expected time.

Proof. This is [10, Summary Theorem]. The situation there is more abstract, more involved. We offer this tip to the reader: their $\partial, z, \sigma, \delta$ correspond respectively to our x, y, identity, identity.

Item (c) is not stated explicitly in a theorem of [10] but is evident from the proofs of Theorems 5.2 and 5.6 there. The common denominator is the determinant of a matrix over $K[z]$ of dimension n^2d and with entries of degree in z at most e. □

We will generalize this fact to nonsingular and non-square matrices in Theorem 5.5. In that case the unimodular cofactor, $\boldsymbol{U}$, is not unique and may have arbitrarily

large degree entries. The following algorithm is designed to produce a $\boldsymbol{U}$ with bounded degrees.

Algorithm 6 $\boldsymbol{U}$, $\boldsymbol{H}$ = `HermiteForm(A)`

Input: Nice matrix $\boldsymbol{A} \in F[y,x]^{m\times n}$, for field F and parameters x, y.

Output: For $K = F(y)$, Unimodular $\boldsymbol{U} \in K[x]^{m\times m}$ and $\boldsymbol{H} \in K[x]^{m\times n}$ in triangular Smith form such that $\boldsymbol{UA} = \boldsymbol{H}$. The point of the specific method given here is to be able, in Theorem 5.5, to bound $\deg_x(\boldsymbol{U},\boldsymbol{H})$ and $\deg_y(\boldsymbol{U},\boldsymbol{H})$ (numerators and denominators).

1: Compute $r = \operatorname{rank}(\boldsymbol{A})$ and nonsingular $\boldsymbol{U}_0 \in K^{m\times m}$ such that $\boldsymbol{A} = \boldsymbol{U}_0\boldsymbol{A}$ has nonsingular leading $r\times r$ minor. Because $\boldsymbol{A}$ is nice the first r columns are independent and such $\boldsymbol{U}_0$ exists. $\boldsymbol{U}_0$ could be a permutation found via Gaussian elimination, say, or a random unit upper triangular matrix. In the random case, failure to achieve nonsingular leading minor becomes evident in the next step, so that the randomization is Las Vegas.

2: Let $\boldsymbol{U}_0\boldsymbol{A} = \begin{bmatrix}\boldsymbol{A}_1 & \boldsymbol{A}_2\\ \boldsymbol{A}_3 & \boldsymbol{A}_4\end{bmatrix}$ and $\boldsymbol{B} = \begin{bmatrix}\boldsymbol{A}_1 & \boldsymbol{0}_{r\times m-r}\\ \boldsymbol{A}_3 & \boldsymbol{I}_{m-r}\end{bmatrix}$. $\boldsymbol{B}$ is nonsingular. Compute its unique unimodular cofactor $\boldsymbol{U}_1$ and Hermite form $\boldsymbol{T} = \boldsymbol{U}_1\boldsymbol{B} = \begin{bmatrix}\boldsymbol{H}_1 & *\\ 0 & *\end{bmatrix}$.
If $\boldsymbol{H}_1$ is in triangular Smith form, let $\boldsymbol{H} = \boldsymbol{U}_1\boldsymbol{U}_0\boldsymbol{A} = \begin{bmatrix}\boldsymbol{H}_1 & \boldsymbol{H}_2\\ 0 & 0\end{bmatrix}$.
Let $\boldsymbol{U} = \boldsymbol{U}_1\boldsymbol{U}_0$ and return $\boldsymbol{U}$, $\boldsymbol{H}$.
Otherwise go back to step 1 and choose a better $\boldsymbol{U}_0$. With high probability this repetition will not be needed; probability of success increases with each iteration.

Theorem 5.5. *Let F be a field, x, y parameters, and let $\boldsymbol{A}$ be in $F[y,x]^{m\times n}$ of rank r, $\deg_x(\boldsymbol{A}) \le d$, and $\deg_y(\boldsymbol{A}) \le e$. Then, for the triangular Smith form form $\boldsymbol{UAR} = H$ computed as $\boldsymbol{U}$, $\boldsymbol{H}$ = `HermiteForm`$(\boldsymbol{AR})$, we have*

(a) Algorithm `HermiteForm` *is (Las Vegas) correct and runs in expected time* $\mathrm{O}(m^7d^3e)$*;*

(b) $\deg_x(\boldsymbol{U},\boldsymbol{H}) \le md$*;*

(c) $\deg_y(\boldsymbol{U},\boldsymbol{H}) = \tilde{O}(m^2de)$.

Proof. Let $\boldsymbol{R}$ be as in Fact 1 with $K = F(y)$ and $S \subset F$. If the field F is small, an extension field can be used to provide large enough S.

We apply `HermiteForm` to $\boldsymbol{AR}$ to obtain $\boldsymbol{U}$, $\boldsymbol{H}$, and use the notation of the algorithm in this proof. We see by construction that $\boldsymbol{B}$ is nonsingular, from which it follows that $\boldsymbol{U}_1$ and $\boldsymbol{T}$ are uniquely determined. $\boldsymbol{B}$ is nice because $\boldsymbol{A}$ is nice and all j-minors of $\boldsymbol{B}$ for $j > r$ are either zero or equal to $\det \boldsymbol{A}_1$. It follows that the leading r columns of $\boldsymbol{H}$ must be those of $\boldsymbol{T}$. The lower left $(m-r)\times(n-r)$ block of $\boldsymbol{H}$ must be zero because $\operatorname{rank}(\boldsymbol{H}) = \operatorname{rank}(\boldsymbol{A})$. The leading r rows are independent, and any nontrivial linear combination of those rows would be nonzero in the lower left block. Then $\boldsymbol{H}$ is in triangular Smith form and left equivalent to $\boldsymbol{A}$ as required. The runtime is dominated by computation of $\boldsymbol{U}_1$ and $\boldsymbol{T}$ for $\boldsymbol{B}$, so Fact 2 provides the bound in (a).

For the degree in x, applying Fact 2, we have $\deg_x(\boldsymbol{U}_1) \le (m-1)d$. Noting that $\boldsymbol{U}_0$ has degree zero, we have $\deg_x(\boldsymbol{U}) = \deg_x(\boldsymbol{U}_1)$ and $\deg_x(\boldsymbol{H}) = \deg_x(\boldsymbol{U}) + \deg_x(\boldsymbol{A}) \le (m-1)d + d = md$.

For the degree in y, note first that the bounds d, e for degrees in $\boldsymbol{A}$ apply as well to $\boldsymbol{B}$. We have, by Fact 2, that $\deg_y(\operatorname{num}(\boldsymbol{U}_1)) = \tilde{O}(m^2de)$ and the same bound for $\deg_y(\operatorname{den}(\boldsymbol{U}_1))$. For $\boldsymbol{H}$, note that $\operatorname{num}(\boldsymbol{H})/\operatorname{den}(\boldsymbol{H}) = \operatorname{num}(\boldsymbol{U})\boldsymbol{A}/\operatorname{den}(\boldsymbol{U})$ so that

and $\deg_y(\text{den}(\boldsymbol{H})) \leq \deg_y(\boldsymbol{U}) = O^{\tilde{}}(m^2de)$, and $\deg_y(\text{num}(\boldsymbol{H})) \leq \deg_y(\text{num}(\boldsymbol{U})A) = O^{\tilde{}}(m^2de) + e = O^{\tilde{}}(m^2de)$. □

5.5 Reduction of PLS to triangular Smith forms

In this section we define the Comprehensive Triangular Smith Normal form problem and solution and show that PLS can be reduced to it. The next section addresses the solution of CTSNF itself.

Definition 5.6. For field F, parameters $Y = (y_1, \ldots, y_s)$, F_Y is a **parameterized extension** of F if $F_Y = F_s$, the top of a tower of extensions $F_0 = F, F_1, \ldots, F_s$ where, for $i \in 1, \ldots, s$, each F_i is either $F_{i-1}(y_i)$ (rational functions) or $F_{i-1}[y_i]/\langle f_i \rangle$, for f_i irreducible in y_i over F_{i-1} (algebraic extension). When a solution regime to a PLS or CTSNF problem is over a parameterized extension F_Y, the irreducible polynomials involved in defining the extension tower for F_Y will be in the constraint set Z of polynomials that must evaluate to zero.

A comprehensive triangular Smith normal form problem (**CTSNF problem**) is a triple $(\boldsymbol{A}, N, Z)$ of a matrix $\boldsymbol{A}$ over $F[Y, x]$ and polynomial sets $N, Z \subset F[Y, x]$, so that $V(N, Z)$ constrains the range of desired parameter values as in the PLS problem.

For CTSNF problem $(\boldsymbol{A}, N, Z)$ over $F[Y, x]$, a **triangular Smith regime** is of the form $(\boldsymbol{U}, \boldsymbol{H}, \boldsymbol{R}, N', Z')$, with $\boldsymbol{U}$, $\boldsymbol{H}$ over $F_Y[x]$, where F_Y is a parameterized extension of F and any polynomials defining algebraic extensions in the tower are in Z', such that on all $a \in V(N', Z')$, $H(a)$ is in triangular Smith form over $F(a)[x]$, $\boldsymbol{U}(a)$ is unimodular in x, $\boldsymbol{R}$ is nonsingular over F, and $\boldsymbol{U}(a)\boldsymbol{A}(a)\boldsymbol{R} = \boldsymbol{H}(a)$.

A **CTSNF solution** is a list $\{(\boldsymbol{U}_i, \boldsymbol{H}_i, \boldsymbol{R}_i, N_i, Z_i) | i \in 1, \ldots, k\}$, of **triangular Smith regimes** that cover $V(N, Z)$, which is to say $V(N, Z) \subset \cup\{V(N_i, Z_i) | i \in 1, \ldots, k\}$.

The goal in this section is to reduce the PLS problem to the CTSNF problem. The first step is to show it suffices to consider PLS with a matrix already in triangular Smith form. The second step is to show each CTSNF solution regime generates a set of PLS solution regimes.

Lemma 5.7. *Given a parameterized field F_Y and matrix $\boldsymbol{A}$ over $F[Y, x]$, let $\boldsymbol{H}$ be a triangular Smith form of $\boldsymbol{A}$ over $F_Y[x]$, with $\boldsymbol{U}$ unimodular over $F_Y[x]$, and $\boldsymbol{R}$ nonsingular over F such that $\boldsymbol{UAR} = \boldsymbol{H}$. PLS problem $(\boldsymbol{A}, \vec{b}, N, Z)$ over $F[Y, x]$ has solution regimes $(\vec{u}_1, \boldsymbol{B}_1, N_1, Z_1), \ldots, (\vec{u}_s, \boldsymbol{B}_s, N_s, Z_s)$ if and only if PLS problem $(\boldsymbol{H}, \boldsymbol{U}\vec{b}, N, Z)$ has solution regimes $(\boldsymbol{R}^{-1}\vec{u}_1, \boldsymbol{R}^{-1}\boldsymbol{B}_1, N_1, Z_1), \ldots, (\boldsymbol{R}^{-1}\vec{u}_s, \boldsymbol{R}^{-1}\boldsymbol{B}_s, N_s, Z_s)$.*

Proof. Under evaluation at any $a \in V(N, Z)$, $\boldsymbol{U}(a)$ is unimodular and $\boldsymbol{R}$ is unchanged and nonsingular. Thus the following are equivalent.

1. $\boldsymbol{A}(a)\vec{u}(a) = \vec{b}(a)$.
2. $\boldsymbol{U}(a)\boldsymbol{A}(a)\vec{u}(a) = \boldsymbol{U}(a)\vec{b}(a)$.
3. $(\boldsymbol{U}(a)\boldsymbol{A}(a)\boldsymbol{R})(\boldsymbol{R}^{-1}\vec{u}(a)) = \boldsymbol{U}(a)\vec{b}(a)$.

□

Then we have the following algorithm to solve a PLS with the matrix already in triangular Smith form. For simplicity we assume a square matrix, the rectangular case being a straightforward extension.

Algorithm 7 `TriangularSmithPLS`

Input: PLS problem $(\boldsymbol{H}, \vec{b}, N, Z)$, with $\boldsymbol{H} \in K_Y[x]^{n\times n}$ and $\vec{b} \in K_Y[x]^n$, where F_Y is a parameterized extension for parameter list Y and x is an additional parameter, with $N, Z \subset F[Y]$ and $\boldsymbol{H}$ in triangular Smith form.

Output: S, the corresponding PLS solution (a list of regimes).

1: For any polynomial $s(x)$ let sqfr(s) denote the square-free part. Let s_i denote the i-th diagonal entry of $\boldsymbol{H}$, and define $s_0 = 1, s_{n+1} = 0$. Then, for $i \in 0, \ldots, n$, define $f_i = \text{sqfr}(s_{i+1})/\text{sqfr}(s_i)$. Let $\mathcal{I}$ be the set of indices such that f_i has positive degree or is zero.

2: For each $r \in \mathcal{I}$ include in the output S the regime $R = (\vec{u}, \boldsymbol{B}, V)$, where $\vec{u} = (\boldsymbol{H}_r^{-1}\vec{b}_r, 0_{n-r})$, with $\boldsymbol{H}_r$ the leading $r \times r$ submatrix of $\boldsymbol{H}$ and $\vec{b}_r = (b_1, \ldots, b_r)$ and $\boldsymbol{B} = (\vec{e}_{r+1}, \ldots, \vec{e}_n)$. Here $\vec{e}_i$ denotes the i-th column of the identity matrix.

3: Return S.

Lemma 5.8. *Algorithm* TriangularSmithPLS *is correct and generates at most* $\sqrt{2d}$ *regimes, where* $d = \deg(\det(\boldsymbol{H}))$.

Proof. Note that, for each row k, the diagonal entry s_k divides all other entries in the row. Then $\boldsymbol{H}$ has rank r just in case $s_r \neq 0$ and $s_{r+1} = 0$, i.e., in the cases determined in step 1 of the algorithm. The addition of s_r to N and f_r to Z ensures rank r and invertibility of $\boldsymbol{H}_r$. For all evaluation points $a \in V(N, Z)$ satisfying those two additional conditions, the last $n-r$ rows of $\boldsymbol{H}$ are zero. Hence the nullspace $\boldsymbol{B}$ is correctly the last $n-r$ columns of $\boldsymbol{I}_n$. For such evaluation points, the system will be consistent if and only if the corresponding right hand side entries are zero, hence the addition of $b_{r+1}, \ldots, b_n$ to Z. □

Algorithm 8 `PLSviaCTSNF`

Input: A PLS problem $(\boldsymbol{A}, \vec{b}, N, Z)$ over $F[Y, x]$, for parameter list Y and additional parameter x.

Output: A corresponding PLS solution $S = ((\vec{u}_i, \boldsymbol{B}_i, N_i, Z_i) | i \in 1, \ldots, s)$.

1: Over the ring $F(Y)[x]$, Let T solve the CTSNF problem $(\boldsymbol{A}, N, Z)$. T is a set of triangular Smith regimes of form $(\boldsymbol{U}, \boldsymbol{H}, R, N', Z')$. Let $S = \emptyset$.

2: For each Hermite regime $(\boldsymbol{U}, \boldsymbol{H}, R, N', Z')$ in T, using algorithm `TriangularSmithPLS`, solve the PLS problem $(\boldsymbol{H}, \boldsymbol{U}\vec{b}, N', Z')$. Adjoin to S the solution regimes, adjusted by factor $\boldsymbol{R}^{-1}$ as in Lemma 5.7.

3: Return S.

Theorem 5.9. *Algorithm 8 is correct.*

Proof. For every parameter evaluation $a \in V(N, Z)$ at least one triangular Smith regime of T in step 2 is valid. Then, by Lemmas 5.8 and 5.7, step 3 produces a PLS regime covering a. □

5.6 Solving Comprehensive Triangular Smith Normal Form

In view of the reductions of the preceding section, to solve a parametric linear system it remains only to solve a comprehensive triangular Smith form problem. This is difficult in general but we give a method to give a comprehensive solution with polynomially many regimes in the bivariate and trivariate cases.

Theorem 5.10. *Let $\boldsymbol{A} \in F[y,x]^{m\times n}$ of degree d in x and degree e in y, and let N, Z be polynomial sets defining a semialgebraic constraint on y. Then the CTSNF problem $(\boldsymbol{A}, N, Z)$ has a solution of at most $\mathrm{O}(n^2de)$ triangular Smith regimes.*

Proof. If N is nonempty, then at the end of the construction below just adjoin N to the N_* of each solution regime. If Z is nonempty it trivializes the solution to at most one regime: Let $z(y)$ be the greatest common divisor of the polynomials in Z. If z is 1 or is reducible, the condition is unsatisfiable, otherwise return the single triangular Smith regime for $\boldsymbol{A}$ over $F[y]/\langle z(y)\rangle$. Otherwise construct the solution regimes as follows where we will assume the semialgebraic constraints are empty.

First compute triangular Smith form $\boldsymbol{U}_0, \boldsymbol{H}_0, \boldsymbol{R}_0$ over $F(y)[x]$ such that $\boldsymbol{A} = \boldsymbol{U}_0\boldsymbol{H}_0\boldsymbol{R}_0$. This will be valid for evaluations that don't zero the denominators (polynomials in y) of $\boldsymbol{H}_0, \boldsymbol{U}_0$. So set $N_0 = \mathrm{den}(\boldsymbol{U}_0, \boldsymbol{H}_0)$ (or to be the set of irreducible factors that occur in $\mathrm{den}(\boldsymbol{U}_0, \boldsymbol{H}_0)$. Set $Z_0 = \emptyset$ to complete the first regime.

Then for each irreducible polynomial $f(y)$ that occurs as a factor in N_0 adjoin the regime $(\boldsymbol{U}_f, \boldsymbol{H}_f, \boldsymbol{R}_f, N_f = N\backslash\{f\}, Z_f = \{f\})$, that comes from computing the triangular Smith form over $(F[y]/\langle f\rangle)[x]$. From the bounds of Theorem 5.5 we have the specified bound on the number of regimes. □

We can proceed in a similar way when there are three parameters, but must address an additional complication that arises.

Theorem 5.11. *Let $\boldsymbol{A} \in F[z,y,x]^{m\times n}$ of degree d in x and degree e in y, z, and let N, Z be polynomial sets defining a semialgebraic constraint on y and z. Then the CTSNF problem $(\boldsymbol{A}, V(N, Z))$ has a solution of at most $\mathrm{O}(n^4d^2e^2)$ triangular Smith regimes.*

Proof. As in the bivariate case above, we solve the unconstrained case and just adjoin N, Z, if nontrivial, to the semialgebraic condition of each solution regime.

First compute triangular Smith form $\boldsymbol{U}_0, \boldsymbol{H}_0, \boldsymbol{R}_0$ over $F(y,z)[x]$ such that $\boldsymbol{A} = \boldsymbol{U}_0\boldsymbol{H}_0\boldsymbol{R}_0$. This will be valid for evaluations that don't zero the denominators (polynomials in y, z) of $\boldsymbol{H}_0, \boldsymbol{U}_0$. Thus we set $(N_0, Z_0) = (\{\mathrm{den}(\boldsymbol{U}_0, \boldsymbol{H}_0)\}, \emptyset)$ to complete the first regime.

Then for each irreducible polynomial f that occurs as a factor in N_0, if y occurs in f, adjoin the regime $(\boldsymbol{U}_f, \boldsymbol{H}_f, \boldsymbol{R}_f, N_f = N\setminus\{f\}, Z_f = \{f\})$, that comes from computing the triangular Smith form over $(F(z)[y]/\langle f\rangle)[x]$. If y doesn't occur in f, interchange the roles of y, z.

In either case we get a solution valid when f is zero and the solution denominator δ_f is nonzero. This denominator is of degree $\mathrm{O}(n^2de)$ in each of y, z by Theorem 5.5. [It is the new complicating factor arising in the trivariate case.] It is relatively prime to f, so Bézout's theorem [7] in the theory of algebraic curves can be applied: there are at most $\deg(f)\deg(\delta)$ points that are common zeroes of f and δ. We can produce a separate regime for each such (y,z)-point by evaluating $\boldsymbol{A}$ at the point and computing a triangular Smith form over $F[x]$. Summing over the irreducible f dividing the original denominator in N_0 we have $\mathrm{O}((n^2de)^2)$ bounding the number of these denominator curve intersection points. □

Corollary 5.12. *For a PLS with $m\times n$ matrix $\boldsymbol{A}$, $\vec{b}$ an m-vector, and with $\deg_x(\boldsymbol{A}, \vec{b}) \leq d, \deg_y(\boldsymbol{A}, \vec{b}) \leq e, \deg_z(\boldsymbol{A}, \vec{b}) \leq e$, we have*

1. *$\mathrm{O}(m^{1.5}d^{0.5})$ regimes in the PLS solution for the univariate case (domain of $\boldsymbol{A}$, $\vec{b}$ is $F[x]$).*
2. *$\mathrm{O}(m^{2.5}d^{1.5}e)$ regimes in the PLS solution for the bivariate case (domain of $\boldsymbol{A}$, $\vec{b}$ is $F[x,y]$).*

3. $O(m^{4.5}d^{2.5}e^2)$ *regimes in the PLS solution for the trivariate case (domain of* $\boldsymbol{A}$, $\vec{b}$ *is* $F[x,y,z]$*).*

Proof. By Lemma 5.8, each CTSNF regime expands to at most $\sqrt{md}$ PLS regimes. □

5.7 Normal forms and Eigenproblems

Comprehensive Hermite Normal form and comprehensive Smith Normal form are immediate corollaries of our comprehensive triangular Smith form. For Hermite form, just take the right hand cofactor to be the identity, $\boldsymbol{R} = \boldsymbol{I}$, and drop the check for the divisibility condition on the diagonal entries in Algorithm 7. For Smith form one can convert each regime of CTSNF to a Smith regime. Where $\boldsymbol{UAR} = \boldsymbol{H}$ with $\boldsymbol{H}$ a triangular Smith form, perform column operations to obtain $\boldsymbol{UAV} = \boldsymbol{S}$ with $\boldsymbol{S}$ the diagonal of $\boldsymbol{H}$. In $\boldsymbol{H}$ the diagonal entries divide the off diagonal entries in the same row. Subtract multiples of the i-th column from the subsequent columns to eliminate the off diagonal entries. Because the diagonal entries are monic, no new denominator factors arise and $\det(\boldsymbol{V}) = \det(\boldsymbol{R}) \in F$. Thus when $(\boldsymbol{U}, \boldsymbol{H}, \boldsymbol{R}, N, Z)$ is a valid regime in a CTSNF solution for $\boldsymbol{A}$, then $(\boldsymbol{U}, \boldsymbol{S}, \boldsymbol{V}, N, Z)$ is a valid regime for Smith normal form.

It is well known that if $\boldsymbol{A} \in K^{n\times n}$ for field K (that may involve parameters) and λ is an additional variable, then the Smith invariants $s_1, \ldots, s_n$ of $\lambda\boldsymbol{I} - \boldsymbol{A}$ are the Frobenius invariants of $\boldsymbol{A}$ and $\boldsymbol{A}$ is similar to its Frobenius normal form, $\oplus_{i=1}^{n} \boldsymbol{C}_{s_i}$, where $\boldsymbol{C}_s$ denotes the companion matrix of polynomial s. Thus we have comprehensive Frobenius normal form as a corollary of CTSNF, however it is without the similarity transform. It would be interesting to develop a comprehensive Frobenius form with each regimes including a transform.

Parametric eigenvalue problems for $\boldsymbol{A}$ correspond to PLS for $\lambda\boldsymbol{I} - \boldsymbol{A}$ with zero right hand side. Often eigenvalue multiplicity is the concern. The geometric multiplicity is available from the Smith invariants, as for example on the diagonal of a triangular Smith form. Common roots of 2 or more of the invariants expose geometric multiplicity and square-free factorization of the individual invariants exposes algebraic multiplicity. Note that square-free factorization may impose further restrictions on the parameters. Comprehensive treatment of square-free factorization is considered in [18].

5.7.1 Eigenvalue multiplicity example

The following matrix, due originally to a question on sci.math.num-analysis in 1990 by Kenton K. Yee, is discussed in [5]. We change the notation used there to avoid a clash with other notation used here. The matrix is

$$\boldsymbol{Y} = \begin{bmatrix} z^{-1} & z^{-1} & z^{-1} & z^{-1} & z^{-1} & z^{-1} & z^{-1} & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & z & z & z & z & z & z & z \end{bmatrix}.$$

One of the original questions was to compute its eigenvectors. Since it contains a symbolic parameter z, this is a parametric eigenvalue problem which we can turn into a parametric linear system, namely to present the nullspace regimes for $\lambda\boldsymbol{I} - \boldsymbol{Y}$.

Over $F(z)[\lambda]$, after preconditioning, we get as the triangular Smith form diagonal $(1,1,1,1,1,\lambda^2-1,\lambda^2-1,(\lambda^2-1)f(\lambda))$, where $f(\lambda)=\lambda^2-(z+6+z^{-1})\lambda+7$.

Remark 5.13. Without preconditioning, the Hermite form diagonal is instead $(1,1,1,1,\lambda-1,\lambda^2-1,(\lambda^2-1),(\lambda+1)f(\lambda))$.

The denominator of $\boldsymbol{U}$, $\boldsymbol{H}$ is a power of z, so the only constraint is $z=0$ which is already a constraint for the input matrix. We get regimes of rank 5 for $\lambda=\pm1$, rank 7 for λ being a root of f, and rank 8 for all other λ. In terms of the eigenvalue problem, we get eigenspaces of dimension 3 for each of 1, -1 and of dimension 1 for the two roots of $f(\lambda)$.

To explore algebraic multiplicity, we can examine when f has 1 or -1 as a root. When z is a root of $z^2+14\,z+1$, $f(\lambda)$ factors as $(\lambda-1)(\lambda-7)$ and when $z=1$ we have $f(\lambda)=(\lambda+1)(\lambda+7)$. These factorizations may be discovered by taking resultants of f with $\lambda-1$ or $\lambda+1$.

5.7.2 Matrix Logarithm

Theorem 1.28 of [12] states conditions under which the matrix equation $\exp(\boldsymbol{X})=\boldsymbol{A}$ has so-called *primary matrix logarithm* solutions, and under which conditions there are more. If the number of distinct eigenvalues s of $\boldsymbol{A}$ is *strictly less* than the number p of distinct Jordan blocks of $\boldsymbol{A}$ (that is, the matrix $\boldsymbol{A}$ is *derogatory*), then the equation also has so-called *nonprimary* solutions as well, where the branches of logarithms of an eigenvalue λ may be chosen differently in each instance it occurs.

As a simple example of what this means, consider

$$\boldsymbol{A}=\begin{bmatrix} a & 1 \\ 0 & a \end{bmatrix}. \tag{5.7.1}$$

When we compute its matrix logarithm (for instance using the `MatrixFunction` command in Maple), we find

$$\boldsymbol{X}_A=\begin{bmatrix} \ln(a) & a^{-1} \\ 0 & \ln(a) \end{bmatrix}. \tag{5.7.2}$$

This is what we expect, and taking the matrix exponential (a single-valued matrix function) gets us back to $\boldsymbol{A}$, as expected. However, if instead we consider the derogatory matrix

$$\boldsymbol{B}=\begin{bmatrix} a & 0 \\ 0 & a \end{bmatrix} \tag{5.7.3}$$

then its matrix logarithm as computed by `MatrixFunction` is also derogatory, namely

$$\boldsymbol{X}_B=\begin{bmatrix} \ln(a) & 0 \\ 0 & \ln(a) \end{bmatrix}. \tag{5.7.4}$$

Yet there are other solutions as well: if we add $2\pi i$ to the first entry and $-2\pi i$ to the second logarithm, we unsurprisingly find another matrix $\boldsymbol{X}_C$ which also satisfies $\exp(\boldsymbol{X})=\boldsymbol{B}$. But adding $2\pi i$ to the first entry of $\boldsymbol{X}_A$ while adding $-2\pi i$ to its second logarithm, we get another matrix

$$\boldsymbol{X}_D=\begin{bmatrix} \ln(a)+2\,i\pi & a^{-1} \\ 0 & \ln(a)-2\,i\pi \end{bmatrix} \tag{5.7.5}$$

which has the (somewhat surprising) property that $\exp(\boldsymbol{X}_D) = \boldsymbol{B}$, not $\boldsymbol{A}$.

This example demonstrates in a minimal way that the detailed Jordan structure of $\boldsymbol{A}$ strongly affects the nature of the solutions to the matrix equation $\exp(\boldsymbol{X}) = \boldsymbol{A}$. This motivates the ability of code to detect automatically the differing values of the parameters in a matrix that make it derogatory. To explicitly connect this example to CTSNF, consider

$$\boldsymbol{M} = \begin{bmatrix} a & b \\ 0 & a \end{bmatrix} \tag{5.7.6}$$

so that $\boldsymbol{A}$ above is $\boldsymbol{M}_{b=1}$ and $\boldsymbol{B} = \boldsymbol{M}_{b=0}$. The CTSNF applied to $\lambda\boldsymbol{I} - \boldsymbol{M}$ produces two regimes, with forms

$$\boldsymbol{H}_{b\neq 0} = \begin{bmatrix} 1 & \lambda/b \\ 0 & (\lambda-a)^2 \end{bmatrix}, \boldsymbol{H}_{b=0} = \begin{bmatrix} \lambda-a & 0 \\ 0 & \lambda-a \end{bmatrix}, \tag{5.7.7}$$

exposing when the logarithms will be linked or distinct. Note that in this case the Frobenius structure equals the Jordan structure.

5.7.3 Model of infectious disease vaccine effect

Rahman and Zou [22] have made a model of vaccine effect when there are two subpopulations with differing disease susceptibility and vaccination rates. Within this study stability of the model is a function of the eigenvalues of a Jacobian $\boldsymbol{J}$. Thus we are interested in cases where the following matrix is singular.

$$\boldsymbol{A} = \lambda\boldsymbol{I} - \boldsymbol{J} = \begin{bmatrix} \lambda-w & 0 & -a & -c \\ 0 & \lambda-x & -b & -d \\ 0 & 0 & \lambda-a-y & c \\ 0 & 0 & b & \lambda-d-z \end{bmatrix}.$$

Here w, x are vaccination rates for the two populations, y, z are death rates, a, d are within population transmission rates, and b, c are the between population transmission rates. We have simplified somewhat: for instance a, b, c, d are transmission rates multiplied by other parameters concerning population counts. Stability depends on the positivity of the largest real part of an eigenvalue. For the sake of reducing expression sizes in this example we will arbitrarily set $y = z = 1/10$. For the same reason we will skip right multiplication by an R to achieve triangular Smith form. Hermite form $\boldsymbol{H}$ of $\lambda\boldsymbol{I} - \boldsymbol{J}$ will suffice, revealing the eigenvalues that are wanted.

$$\boldsymbol{H} = \begin{bmatrix} \lambda+w & 0 & 0 & -(ad - a\lambda - bc - a/10)/c \\ 0 & \lambda+x & 0 & \lambda + 1/10 \\ 0 & 0 & 1 & (d - \lambda - 1/10)/c \\ 0 & 0 & 0 & \lambda^2 + (1/5 - a - d)\lambda + ad - cb - (1/10)d - (1/10)a + 1/100 \end{bmatrix}.$$

The discriminant of the last entry gives the desired information for the application subject to the denominator validity: $c \neq 0$. When $c = 0$ the matrix is already in Hermite form, so again the desired information is provided.

This example illustrates that often more than three parameters can be easily handled. In experiments with this model not reported here, we did encounter cases demanding solution beyond the methods of this paper. On a more positive note, we feel that comprehensive normal form tools could help analyze models like this when larger in scope, for instance modeling 3 or more subpopulations.

5.7.4 The Kac-Murdock-Szegö example

In [4] we see reported times for computation of the comprehensive Jordan form for matrices of the following form, taken from [28], of dimensions 2 to about 20:

$$\mathrm{KMS}_n = \begin{bmatrix} 1 & -\rho & & & \\ -\rho & \rho^2+1 & -\rho & & \\ & \ddots & \ddots & \ddots & \\ & & -\rho & \rho^2+1 & -\rho \\ & & & -\rho & 1 \end{bmatrix}. \tag{5.7.8}$$

This is, apart from the $(1,1)$ entry and the (n,n) entry, a Toeplitz matrix containing one parameter, ρ. The reported times to compute the Jordan form were plotted in [4] on a log scale, and looked as though they were exponentially growing with the dimension, and were reported in that paper as growing exponentially.

The theorem of this paper states instead that polynomial time is possible for this family, because there are only two parameters (ρ and the eigenvalue parameter, say λ). The Hermite forms for these matrices are all (as far as we have computed) trivial, with diagonal all 1 except the final entry which contains the determinant. Thus all the action for the Jordan form must happen with the discriminant of the determinant. Experimentally, the discriminant with respect to λ has degree n^2+n-4 for KMS matrices of dimension $n \geq 2$ (this formula was deduced experimentally by giving a sequence of these degrees to the Online Encyclopedia of Integer Sequences [26]) and each discriminant has a factor $\rho^{n(n-1)}$, leaving a nontrivial factor of degree $2n-4$ growing only linearly with dimension. The case $\rho = 0$ does indeed give a derogatory KMS matrix (the identity matrix). The other factor has at most a linearly-growing number of roots for each of which we expect the Jordan form of the corresponding KMS matrix to have one block of size two and the rest of size one. We therefore see only polynomial cost necessary to compute comprehensive Jordan forms for these matrices, in accord with our theorem.

5.8 Conclusions

We have shown that using the CTNSF to solve parametric linear systems is of cost polynomial in the dimension of the linear system and polynomial in parameter degree, for problems containing up to three parameters. This shows that polynomially many regimes suffice for problems of this type. To the best of our knowledge, this is the first method to achieve this polynomial worst case.

It remains an open question whether, for linear systems with a fixed number of parameters greater than three, a number of regimes suffices that is polynomial in the input matrix dimension and polynomial degree of the parameters, being exponential only in the number of parameters.

Through experiments with random matrices we have indication that the worst case bounds we give are sharp, though we haven't proven this point. As the examples indicated, many problems will have fewer regimes, and sometimes substantially fewer regimes. We have not investigated the effects of further restrictions of the type of problem, such as to sparse matrices.

Acknowledgements. This work was supported by the Natural Sciences and Engineering Research Council of Canada and by the Ontario Research Centre for Computer

Algebra. The third author, L. Rafiee Sevyeri, would like to thank the Symbolic Computation Group (SCG) at the David R. Cheriton School of Computer Science of the University of Waterloo for their support while she was a visiting researcher there.

Bibliography

[1] Boyer, B., Kaltofen, E.L.: Numerical linear system solving with parametric entries by error correction. In: Proceedings of the 2014 Symposium on Symbolic-Numeric Computation. pp. 33–38 (2014)

[2] Buchberger, B.: Applications of gröbner bases in non-linear computational geometry. In: Janßen, R. (ed.) Trends in Computer Algebra. pp. 52–80. Springer Berlin Heidelberg, Berlin, Heidelberg (1988)

[3] Camargos Couto, A.C., Moreno Maza, M., Linder, D., Jeffrey, D.J., Corless, R.M.: Comprehensive lu factors of polynomial matrices. In: Slamanig, D., Tsigaridas, E., Zafeirakopoulos, Z. (eds.) Mathematical Aspects of Computer and Information Sciences. pp. 80–88. Springer International Publishing, Cham (2020)

[4] Corless, R.M., Moreno Maza, M., Thornton, S.E.: Jordan canonical form with parameters from Frobenius form with parameters. In: Blömer, J., Kotsireas, I.S., Kutsia, T., Simos, D.E. (eds.) Mathematical Aspects of Computer and Information Sciences. pp. 179–194. Springer International Publishing (2017)

[5] Corless, R.M.: Essential Maple 7: an introduction for scientific programmers. Springer Science & Business Media (2002)

[6] Corless, R.M., Jeffrey, D.J.: The Turing factorization of a rectangular matrix. SIGSAM Bull. **31**(3), 20âĂŞ30 (Sep 1997). https://doi.org/10.1145/271130.271135, https://doi.org/10.1145/271130.271135

[7] Cox, D., Little, J., O'Shea, D.: Ideals, varieties, and algorithms: an introduction to computational algebraic geometry and commutative algebra. Springer Science & Business Media (2013)

[8] Dautray, R., Lions, J.: Mathematical analysis and numerical methods for science and technology. Vol. 2. Springer-Verlag, Berlin (1988). https://doi.org/10.1007/978-3-642-61566-5, https://doi.org/10.1007/978-3-642-61566-5

[9] Dessombz, O., Thouverez, F., Laîné, J.P., Jézéquel, L.: Analysis of mechanical systems using interval computations applied to finite element methods. J. Sound Vibration **239**(5), 949–968 (2001). https://doi.org/10.1006/jsvi.2000.3191, https://doi.org/10.1006/jsvi.2000.3191

[10] Giesbrecht, M., Kim, M.S.: Computing the Hermite form of a matrix of Ore polynomials. J. Algebra **376**, 341–362 (2013)

[11] Goldman, L.: Integrals of multinomial systems of ordinary differential equations. J. Pure Appl. Algebra **45**(3), 225–240 (1987). https://doi.org/10.1016/0022-4049(87)90072-7, https://doi.org/10.1016/0022-4049(87)90072-7

[12] Higham, N.J.: Functions of matrices: theory and computation, vol. 104. Siam (2008)

[13] Jensen, K.: Coloured Petri nets. Vol. 1. Monographs in Theoretical Computer Science. An EATCS Series, Springer-Verlag, Berlin (1997). https://doi.org/10.1007/978-3-642-60794-3, https://doi.org/10.1007/978-3-642-60794-3, basic concepts, analysis methods and practical use, Corrected reprint of the second (1996) edition

[14] Kaltofen, E., Krishnamoorthy, M., Saunders, B.D.: Fast parallel computation of Hermite and Smith forms of polynomial matrices. SIAM Journal on Algebraic Discrete Methods **8**(4), 683–690 (1987)

[15] Kaltofen, E.L., Pernet, C., Storjohann, A., Waddell, C.: Early termination in parametric linear system solving and rational function vector recovery with error correction. In: Proceedings of the 2017 ACM on International Symposium on Symbolic and Algebraic Computation. pp. 237–244 (2017)

[16] Kapur, D., Sun, Y., Wang, D.: An efficient algorithm for computing a comprehensive Gröbner system of a parametric polynomial system. Journal of Symbolic Computation **49**, 27–44 (2013)

[17] Kolev, L.V.: Outer solution of linear systems whose elements are affine functions of interval parameters. Reliab. Comput. **8**(6), 493–501 (2002). https://doi.org/10.1023/A:1021320711392, https://doi.org/10.1023/A:1021320711392

[18] Lemaire, F., Maza, M.M., Xie, Y.: The regularchains library. In: Maple conference. vol. 5, pp. 355–368 (2005)

[19] Montes, A.: The Gröbner Cover. Springer (2018)

[20] Montes, A., Wibmer, M.: Gröbner bases for polynomial systems with parameters. Journal of Symbolic Computation **45**(12), 1391–1425 (2010)

[21] Muhanna, R.L., Mullen, R.L.: Uncertainty in mechanics problemsâĂŤintervalâĂŞbased approach. J. of Engineering Mechanics **127**(6), 557–566 (2001)

[22] Rahman, S.A., Zou, X.: Modelling the impact of vaccination on infectious diseases dynamics. Journal of biological dynamics **9**(sup1), 307–320 (2015)

[23] Savageau, M.A., Voit, E.O., Irvine, D.H.: Biochemical systems theory and metabolic control theory. I. Fundamental similarities and differences. Math. Biosci. **86**(2), 127–145 (1987). https://doi.org/10.1016/0025-5564(87)90007-1, https://doi.org/10.1016/0025-5564(87)90007-1

[24] Savageau, M.A., Voit, E.O., Irvine, D.H.: Biochemical systems theory and metabolic control theory. II. The role of summation and connectivity relationships. Math. Biiosc. **86**(2), 147–169 (1987). https://doi.org/10.1016/0025-5564(87)90008-3, https://doi.org/10.1016/0025-5564(87)90008-3

[25] Sit, W.Y.: An algorithm for solving parametric linear systems. Journal of Symbolic Computation **13**(4), 353–394 (1992)

[26] Sloane, N.: The on-line encyclopedia of integer sequences (2020), http://oeis.org

[27] Storjohann, A.: Algorithms for matrix canonical forms. Ph.D. thesis, ETH Zurich (2000)

[28] Trench, W.F.: Properties of some generalizations of Kac-Murdock-Szego matrices. Contemporary Mathematics **281**, 233–246 (2001)

[29] Wahl, L.M., Betti, M.I., Dick, D.W., Pattenden, T., Puccini, A.J.: Evolutionary stability of the lysis-lysogeny decision: Why be virulent? Evolution **73**(1), 92–98 (2019)

[30] Weispfenning, V.: Comprehensive Gröbner bases. Journal of Symbolic Computation **14**(1), 1–29 (1992)

[31] Winkels, H., Meika, M.: An integration of efficiency projections into the Geoffrion approach for multiobjective linear programming. European J. of Operational Research **16**(1), 113–127 (1984)

Chapter 6

Concluding Remarks

We have used common numerical and symbolic methods such as rootfinding, eigenvalue based methods in this thesis and addressed problems related to matrices, polynomials and matrix polynomials in multiple bases.

In order to show relations between different chapters, we present the following example.

Consider Wilkinson polynomial 1.2.10 of degree 5 divided by its largest coefficient, 274. Our goal is to demonstrate that W_5 is quite sensitive. In other words, making a small random change in the polynomial will make it have a root in common with another random polynomial.

Assume f and g are random polynomials of degree 5 and we divide them by their largest coefficient. So we have

$$W_5(z) = \frac{(z-1)(z-2)(z-3)(z-4)(z-5)}{274}$$
$$f(z) = \frac{22\,z^5 - 55\,z^4 - 94\,z^3 + 87\,z^2 - 56\,z}{87}$$
$$g(z) = \frac{-62\,z^5 + 97\,z^4 - 73\,z^3 - 4\,z^2 - 83\,z - 10}{97}$$

Find the Bézout matrix of $B = W_5 + t * f$ and g with respect to z. This gives a matrix polynomial in t:

$$B = \begin{bmatrix} -\frac{3725}{13289} - \frac{220\,t}{8439} & \frac{5895}{13289} + \frac{550\,t}{8439} & -\frac{4805}{13289} + \frac{940\,t}{8439} & \frac{885}{13289} - \frac{10\,t}{97} & -\frac{6350}{13289} + \frac{560\,t}{8439} \\ \frac{16905}{26578} - \frac{1766\,t}{2813} & -\frac{32783}{26578} + \frac{3259\,t}{2813} & \frac{24737}{26578} + \frac{4264\,t}{8439} & \frac{10161}{26578} - \frac{6505\,t}{8439} & \frac{885}{13289} - \frac{10\,t}{97} \\ -\frac{6977}{13289} + \frac{5306\,t}{8439} & \frac{19395}{13289} - \frac{13517\,t}{8439} & -\frac{24774}{13289} + \frac{16504\,t}{8439} & \frac{24737}{26578} + \frac{4264\,t}{8439} & -\frac{4805}{13289} + \frac{940\,t}{8439} \\ \frac{5197}{26578} - \frac{2478\,t}{2813} & -\frac{10552}{13289} + \frac{18439\,t}{8439} & \frac{19395}{13289} - \frac{13517\,t}{8439} & -\frac{32783}{26578} + \frac{3259\,t}{2813} & \frac{5895}{13289} + \frac{550\,t}{8439} \\ -\frac{833}{26578} - \frac{44\,t}{291} & \frac{5197}{26578} - \frac{2478\,t}{2813} & -\frac{6977}{13289} + \frac{5306\,t}{8439} & \frac{16905}{26578} - \frac{1766\,t}{2813} & -\frac{3725}{13289} - \frac{220\,t}{8439} \end{bmatrix}$$

Polynomial eigenvalues of B are

$$-0.022 - 0.013i,\ \ -0.022 + 0.013i,\ \ 0.50 - 0.64i,\ \ 0.50 + 0.64i,\ \ 6.13.$$

The Polynomial eigenvalues of B tells us when $W_5 + tf$ and g have common roots. The methods of this thesis can be used: from Chapter 2 (express everything in Bernstein basis), from Chapter 3 (express everything in Lagrange basis), or Chapter 5.

Another example borrowed from [1, P. 68-71] is to consider

$$S = \begin{bmatrix} 130\,t - 149 & -390\,t - 50 & -154 \\ 43\,t + 537 & -129\,t + 180 & 546 \\ 133\,t - 27 & -399\,t - 9 & -25 \end{bmatrix}.$$

When $t = 0$ this is the matrix gallery(3) in MATLAB, which is a famous example with ill-conditioned eigenvalues (which are 1, 2, and 3). Looking at the above matrix one can find a small value of t, namely about 10^{-6}, for which the matrix S has multiple eigenvalues. The PLS algorithm introduced in Chapter 5 can discover that information.

Hybrid symbolic-numeric methods are introduced in this thesis which address approximate `GCD` problem in Bernstein and Lagrange bases which avoids ill-conditioned basis conversions. The algorithms use companion pencils for computing roots and then follow Victor Pan's algorithm with modifications. An important part of our algorithm is clustering the roots of polynomials. We have introduced multiple deterministic and heuristic algorithms for clustering. Improvement of clustering algorithms is a potential interesting problem for future work. Moreover, generalizing the same ideas to multivariate `GCD` is another open problem related to approximate `GCD` problem which is studied in chapters 2 and 3. The algorithm given in Chapter 3 which computes the approximate `GCD` in Lagrange bases can be trivially extended to Hermite basis. The only difference will be the rootfinding method. In order to compute the roots one can use the companion pencil presented in 4.3.4.

For a given parametric linear system with at most three parameters (in the entries of the matrix), we have provided a method for finding solutions in terms of regimes. Our method is more symbolic rather than numerical approach. The cost of our algorithm is polynomial in the dimension of the coefficient matrix and degree of polynomials appear in the matrix. A natural interesting generalization of the algorithm introduced in Chapter 5, is to find an algorithm to solve a parametric linear system with more than three parameters with polynomial complexity (for any fixed number of parameters).

Bibliography

[1] R. M. Corless. Essential Maple 7: an introduction for scientific programmers. *Springer Science & Business Media*, 2002.

Chapter 7

Curriculum Vitae

Leili Rafiee Sevyeri

Department of Computer Science , MC 327D
Western University
London, Ontario
Canada lrafiees [*at*]uwo.ca
David R. Cheriton School of Computer Science , SCG 2302E
University of Waterloo
Waterloo, Ontario
Canada lrafiees [*at*]uwaterloo.ca

Education

Western University, London, Canada
Ph.D., Applied Mathematics, 2016- now.
Supervisor:
Prof. Robert M. Corless

Waterloo University, Waterloo, Canada
Visiting Researcher, David R. Cheriton School of Computer Science, 2018- now.
Supervisor:
Prof. George Labahn

Western University, London, Canada
M.S., Applied Mathematics, 2016.

University of Guilan, Rasht, Iran
M.S., Pure Mathematics, 2011.

University of Guilan, Rasht, Iran
B.S., Pure Mathematics, 2009.

Research

Computer Algebra, Numerical Analysis, Hybrid Symbolic-Numeric Computation

Teaching

David R. Cheriton School of Computer Science,
University of Waterloo, Waterloo, Canada
Instructor, Introduction to Computational Mathematics (CS371), 2020.

Department of Mathematics,
Western University, London, Canada
Teaching Assistant, Linear Algebra with Numerical Analysis
For Engineering, 2015-2016.

Teaching Assistant, Calculus, 2014-2019.

Teaching Assistant, Advanced Calculus, 2018.

Teaching Assistant, Applied Math and Numerical Methods, 2014.

Huron College, London, Canada
Teaching Assistant, Calculus, 2017-2018.

Teaching Assistant, Methods of matrix algebra, 2017-2018.

Teaching Assistant, Methods of finite mathematics, 2018.

Publications

Robert M. Corless and Leili Rafiee Sevyeri,
Approximate `GCD` in a Bernstein basis.
Maple in Mathematics Education and Research.
Springer International Publishing, 77–91 (2020).

Robert M. Corless and Leili Rafiee Sevyeri,
The Runge Example for Interpolation and Wilkinson's Examples for Rootfinding
SIAM Review, **62**(1), 231–243 (2020).

Robert M. Corless and Leili Rafiee Sevyeri,
Approximate `GCD` in Lagrange bases.
International Symposium on Symbolic and Numeric Algorithms
for Scientific Computing, SYNASC (2020).

Robert M. Corless, Mark Giesbrecht, Leili Rafiee Sevyeri and B. David Saunders
On Parametric Linear System Solving.
Computer Algebra in Scientific Computing, CASC (2020).

Robert M. Corless and Leili Rafiee Sevyeri,
Stirling's Original Asymptotic Series from a Formula like one of Binet's and
its Evaluation by Sequence Acceleration,
Experimental Mathematics, Taylor & Francis, 1–8 (2019).

Robert M. Corless and Leili Rafiee Sevyeri,
Approximate `GCD` in a Bernstein basis,
Poster presented at: ISSAC 2019, The 44th International Symposium on Symbolic
and Algebraic Computation, July 2019, Beijing, China.

Eunice Y. S. Chan, Robert M. Corless, Leili Rafiee Sevyeri
Generalized Standard Triples for Algebraic Linearizations of Matrix Polynomials
arXiv:1805.04488, math.NA (2018).

Robert M. Corless and Leili Rafiee Sevyeri,
Linearization of A Specific Family of Bèzout Matrices.
ACM Commun. Comput. Algebra, **51**, 21–22 (2017).

Leili Rafiee Sevyeri,
A Sequence of Symmetric Bèzout Matrix Polynomials (Master Thesis),
Western University (2016).

Awards

Distinguished Poster Award
The 44th International Symposium on Symbolic and Algebraic Computation
ISSAC 2019, Beijing, China, 2019.

Distinguished Poster Award
The 41st International Symposium on Symbolic and Algebraic Computation
ISSAC 2016, Waterloo, Canada, 2016.

Honored and rewarded as Outstanding (Exceptional Talent) Student
Department of Mathematics, University of Guilan, Rasht, Iran 2008.