Dissertation

Preconditioning polynomial systems using Macaulay dual spaces

Submitted by

Steven L. Ihde

Department of Mathematics

In partial fulfillment of the requirements

For the Degree of Doctor of Philosophy

Colorado State University

Fort Collins, Colorado

Summer 2015

Doctoral Committee:

Advisor: Daniel J. Bates

Chris Peterson
Alexander Hulpke
Peter Young

Abstract

preconditioning polynomial systems using macaulay dual spaces

Polynomial systems arise in many applications across a diverse landscape of subjects. Solving these systems has been an area of intense research for many years. Methods for solving these systems numerically fit into the field of numerical algebraic geometry. Many of these methods rely on an idea called homotopy continuation. This method is very effective for solving systems of polynomials in many variables. However, in the case of zero-dimensional systems, we may end up tracking many more solutions than actually exist, leading to excess computation. This project preconditions these systems in order to reduce computation. We present the background on homotopy continuation and numerical algebraic geometry as well as the theory of Macaulay dual spaces. We show how to turn an algebraic geometric preconditioning problem into one of numerical linear algebra. Algorithms for computing an H-basis and thereby preconditioning the original system to remove extraneous calculation are presented. The concept of the Closedness Subspace is introduced and used to replace a bottleneck computation. A novel algorithm employing this method is introduced and discussed.

Table of Contents

CHAPTER 1

Introduction

Preconditioning is a word often used in the world of Mathematics. The term implies some sort of preprocessing done in order to reduce computation time or create more accurate results. The context here relates to restructuring a system of polynomial equations, without compromising their solution set, in order to more effectively run homotopy continuation algorithms. It is the goal of this project to reduce the computation time for finding the solutions to zero-dimensional polynomial systems by homotopy continuation. We accomplish this by removing extraneous pieces of the algebraic variety associated with the ideal of the polynomial system. As will be seen in Chapter 2, we may end up with a large number of solutions diverging to ∞. If we can efficiently remove these solutions before computation, we have saved computation time. The method for doing this involves finding a better basis for the ideal generated by the system of polynomials.

First, we must homogenize the system of equations. This increases the number of variables by one and moves the system into projective space. The homogenizing variable, used to make each of the monomials the same degree, plays the role of ∞ while in projective space. By using an ideal quotient operation, we can remove degrees of the homogenizing variable. At each step in the computation, we are removing at least one degree of the homogenizing variable in the system. This corresponds to the Zariski closure of the set subtraction of the corresponding varieties. In other words, this will allow us to geometrically remove the pieces of the solution set living at ∞. To do this, we will reduce the problem from one in algebraic geometry to one of numerical linear algebra.

The next step is to represent the homogenized polynomial system as a matrix in order to take advantage of the properties of the Macaulay dual space. Once in the dual space, we can manipulate the ideal associated with the polynomial system without changing the corresponding algebraic variety. We do this through calculation of an *H-basis* as a better basis for the ideal of the polynomial system. Then, we will show that this H-basis has removed the infinite solutions by quotienting out by the homogenizing variable. The result is an instance of homotopy continuation with few to no infinite paths, allowing for faster and smoother computations.

This monograph will provide background on the foundations of numerical algebraic geometry, necessary numerical linear algebra, h-bases and Macaulay dual spaces. We present the current state of algorithms as well as a survey of different algorithms previously used for similar computations. This will include previous successful algorithms as well as new algorithmic changes. The most recent work is a proof of concept that this method can take advantage of the closedness subspace algorithm in order to remove a bottleneck. The background will be given in Chapter 2. Explicit algorithms and examples are provided in Chapters 2 and 3 respectively.

CHAPTER 2

Background

2.1. Numerical Algebraic Geometry

Numerical algebraic geometry is the area of research concerned with solving systems of polynomials using numerical techniques. This is opposed to symbolic methods which were first introduced in the 1960s. Symbolic methods are very fast for small to medium systems with exact coefficients. However, these methods run into computational trouble as the number of variables and the number of equations grows. Unless the system is very highly structured, the number of necessary computations grows combinatorially. The methods are unable to effectively take advantage of modern architecture as they are not generally parallelizable and do not easily allow for inexact coefficients. In order to take advantage of parallel architecture and allow for inexact coefficients, we need numerical tools. The main technique involved in the area of numerical algebraic geometry is that of homotopy continuation. While there are many different approaches within the family of homotopy continuation methods, the main idea is always the same.

We start with a system of equations, $\mathcal{F}$, with coefficients in an algebraically closed field, $\mathbb{C}$, in variables $\bar{x}$, for which we wish to find the solutions. For our purposes we will always use $\mathbb{C}$ and $\bar{x} = x_1, \ldots, x_n$. Next, we create a second system, a *start system* $\mathcal{G}$, with the same or possibly a greater number of solutions which can be easily solved. Since can solve the start system, we have a set of initial solutions to a system in the same family as the target system from which to work. The choosing and solving of the start system will be discussed in more detail later in this chapter. Once we have a start system, we can then build a homotopy, $\mathcal{H}(\bar{x}, t) = (1-t)\mathcal{F}(\bar{x}) + t\mathcal{G}(\bar{x})$ with $t \in [0, 1]$. This function in the variables $\{\bar{x}, t\}$ creates a

continuous morphing from the solutions of the system $\mathcal{G}$ to the solutions of the system $\mathcal{F}$. At time $t = 1$ we have the solutions to the system $\mathcal{G}$ and at time $t = 0$, we end with the solutions to the system $\mathcal{F}$. These solutions may be unique or they may have a multiplicity greater than one. There is also the possibility that we may get higher dimensional components or have solutions that diverge to ∞.

For the purposes of numerical algebraic geometry, we want the solutions to polynomial systems. These systems have algebraic structure that allows us to exploit deep theorems. However, these homotopy methods can be applied to any nonlinear system given the correct considerations. My research focuses on homotopy continuation using the software *Bertini* [2]. There are also software packages that use polyhedral homotopies which are outside the scope of this paper and will only be briefly mentioned. These include *PHCpack* [24] and *Hom4PS-2.0* [16].

2.1.1. Predictor/Corrector Methods. Once we have created the homotopy $\mathcal{H}(\bar{x}, t)$ and the solutions to our start system, $\mathcal{G}$, we then have to track the solutions. Starting with the solutions to our start system $\mathcal{G}$, we want to follow the solutions as we vary t from 0 to 1 in the homotopy. Since the homotopy is a continuous deformation, the solutions create a "path" which we can approximate and follow. We do this via a series of discrete time steps. As we track from $t = 1$ to $t = 0$ we predict the direction of the solution path using Euler's method or a higher order predictor. We then stop time and use Newton's method to correct back down to the solution. There are many layers to this calculation and different choices available for the predictor portion.

Definition 2.1.1. *Given a system of polynomial equations* $\mathcal{F} = f_1, \ldots, f_n \in \mathbb{C}[x_1, \ldots, x_n]$ *the* ***Jacobian*** *of* $\mathcal{F}$*,* $J_{\mathcal{F}}$ *is the matrix of partial derivatives where the* i, j *entry is* $\frac{\partial f_i}{\partial x_j}$

DEFINITION 2.1.2. *Given a matrix, M, and the singular values σ_i of M, the* ***Condition Number*** *of M is the ratio of the largest singular value to the smallest singular value, i.e. $\frac{\sigma_{max}}{\sigma_{min}}$, or ∞ if $\sigma_{min} = 0$.*

A particular concern that comes up in homotopy continuation is the question of how much precision to use. If, at each time step, the Newton iteration converges very quickly and the condition number of the Jacobian matrix is within tolerances, then the amount of precision used can be fairly low. However, if the solution path nears a singularity, the condition number of the Jacobian increases tremendously and the amount of precision needed to accurately keep track of the solution increases. This occurs because, as the path nears the singularity, the smallest singular value gets very small. Since this is the denominator of the condition number, it causes the condition number to head towards infinity. For this, we use adaptive multiprecision [4], which adapts the amount of precision. At each step, if the condition number is above a proscribed tolerance, then Bertini will increase the precision and recalculate the step. If, during successive steps, the condition number drops, then Bertini can adapt the precision back down as necessary.

Another consideration is how far to step in time. If we take too large a step, it is possible that we will miss and jump into the Newton convergence zone of a different solution. In this situation, we would appear to miss a solution and have a higher multiplicity on another solution. On the other hand, if we take too small a step, we may end up unnecessarily taking too many steps for something that is fairly smooth. This would cause a tremendous amount of extraneous computation. The standard solution to that problem is to use adaptive step length [3]. This allows a predetermined step size to be used, then adapted to smaller step sizes as needed. If the Newton iteration does not converge quickly enough and the condition

number increases, the step size will be adjusted and the prediction reevaluated. The step sizes can then lengthen back to the original size as conditions improve.

2.1.2. Start Systems. One concern for homotopy continuation methods is how to create a start system. We need a system that has the same number of solutions, or possibly a greater number of solutions, which is easily solved. There are many different methods for doing this. The most basic is called a Total Degree homotopy. We know that the system of polynomials can have at most the *Bézout number* of zero-dimensional solutions. The Bézout bound is the product of the degrees of the polynomials and is an upper bound on the number of isolated solutions. The Total Degree homotopy includes a start system consisting of equations for the roots of unity in each variable. Since the roots of unity are an easily known set of solutions, creating and solving this start system is very fast. Unfortunately in many cases, especially sparse systems, the Bézout bound is not a very tight bound. So, using a Total Degree homotopy, we are able to create a very easy start system, but this may cause extra cost in tracking many extraneous paths. It is this particular start system with which this work is most concerned.

There are many other types of start systems to use. Much research in this area has focused on new ways to build start systems in order to reduce the number of paths. After total degree, multi-homogeneous and linear products are the next most difficult start systems to build and solve, respectively. Beyond that, we have monomial products, polynomial products, and mixed volume calculations. The mixed volume calculation relates to the polyhedral homotopy and involves calculating the volumes of newton polytopes. The most difficult start system is the parameter-coefficient homotopy as this requires the most up front work to build and solve the start system. Once the parameter homotopy start system is solved,

though, it allows for fast computations over a large number of parameters, see [8]. For more information on the various types of start systems see [21] and [6].

2.2. H-bases

In linear algebra, it is often the case that we would like to change from one basis of a vector space to another. Similarly in algebraic geometry, we would like to find a nicer basis for our polynomial ideal $\mathcal{I}$. One such formulation of this is that of Gröbner bases. A Gröbner basis is a basis for a polynomial ideal with special properties. The standard process to compute a Gröbner basis requires, first, that we have a monomial ordering. This ensures that any two distinct monomials can be put in order and we never have a tie. Secondly, it is necessary that each of the basis elements is "divisible" by all others in the ideal according to the ordering. This extension of the division algorithm is a necessary condition in order to form the Gröbner basis. With these things in hand, there is an algorithm for finding a Gröbner basis due to Buchberger, see [10].

Unfortunately, under certain conditions, Gröbner basis computations will fail. If the coefficients of the polynomials are not given exactly, these methods break down. Even with exact coefficients, there are computational problems with Gröbner bases. The first is that the combinatorial nature of the division algorithm means that these methods require too much memory once the number of equations and variables gets too high. The only way to combat this is to work with systems that have large amounts of structure. For very sparse systems, the lack of symmetry and structure causes issues. All of the possible monomials are required to be checked, even if they don't contribute to the system. The second problem is that, since the computation is highly serial, there is not yet a good way to parallelize these methods in order to take advantage of multi-core technology. While some parts of the

computation can be parallelized, the main bottleneck remains necessarily serial due to the structure of the algorithm. Both of these problems are created by the division criteria. The necessity that each of the basis elements be divisible means that as new basis elements are added, each must be checked in turn against all others.

In 1916, F.S. Macaulay introduced the idea of an H-basis [17]. This is a homogeneous basis for an ideal based on a degree condition, as we will see in the formal definition. Macaulay was able to create these bases but his methods were unclear. Due to the computational nature and lack of processing power, the process was not well understood until the late 1990's. The advent of advanced computers and computing power has made these computations feasible. H-bases are numerically stable; they are not sensitive to exact coefficients [18]. So, unlike Gröbner bases, they still work with inexact coefficients. Also, h-bases do not require a division algorithm and therefore do not have the combinatorial issues that result from such a condition. The only condition will be a degree condition, marked as (2) in the following definition.

DEFINITION 2.2.1. *Consider a system of polynomials* $\mathcal{F} = \{f_1, \ldots, f_s\}$ *with* $f_i \in \mathbb{C}[x_1, \ldots, x_n]$. *Let* $G = \{g_1, \ldots, g_r\}$ *be a finite set of homogeneous polynomials with* $g_i \in \mathbb{C}[x_0, \ldots, x_n] \setminus \{0\}$. *We say* G *is an* ***H-Basis*** *for the ideal* $I = < f_1, \ldots, f_s >$ *if for all* $p \in I$, *there exists* $h_1, \ldots, h_r$ *such that*

(1) $p = \sum_{i=1}^{r} h_i g_i$ ***and***

(2) $(deg(h_i) + deg(g_i)) \leq deg(p) \quad$ *for* $i = 1, \ldots, r$

Under these conditions, we have a homogeneous basis for our polynomial ideal. More important to this theory, though, will be the method for which we reach such a basis. In

the next section, we will see a construct by which we will be able to generate an H-basis for zero-dimensional polynomial ideals.

2.3. Macaulay Dual Spaces

From linear algebra, we will incorporate the concept of a dual space. This space, which will be defined shortly, has properties that make it advantageous. It is sometimes the case that a difficult problem can be made simpler by mapping into its dual space. The concept of duality has been applied in many different disciplines. We will be looking at the dual space to a vector space as in the linear algebra definition. In our specific case, we will be able to make a polynomial problem into one of linear algebra. This reduction will create the opportunity to implement numerically stable, highly parallelizable algorithms. From [22] we have,

Definition 2.3.1. *The* ***Dual Space*** V^* *of a vector space* V *over a field* $\mathbb{C}$ *is the vector space of all linear functionals on* V*,* $\quad L : V \to \mathbb{C}$

We want to be able to find a dual space for a system $\mathcal{F}$ of polynomials living in $\mathbb{C}[x_1, \ldots, x_n]$. The ideal $\mathcal{I}(\mathcal{F})$, however, is not a vector space because it is not linear but polynomial. By the nature of the definition of an ideal, each element only needs to be polynomial in the generators. From algebraic geometry, see [9], we have that the set of all homogeneous polynomials forms a graded module, graded in each degree. Since we work over $\mathbb{C}[x_1, \ldots, x_n]$ each graded piece is a vector space. For homogeneous polynomials, this implies that the space of all homogeneous monomials of degree d forms a vector space.

In order to build an appropriate dual space, we need a few definitions. We will start with the differential operators ∂_α. For $\alpha \in \mathbb{Z}^n_{\geq 0}$ we have $|\alpha| = \alpha_1 + \ldots + \alpha_n$ and $\alpha! = \alpha_1!\alpha_2! \cdots \alpha_n!$ Now we are able to define our differential operators ∂_α.

DEFINITION 2.3.2. *The differential operator ∂_α is given as*

$$\partial_\alpha = \frac{1}{\alpha!}\frac{\partial^{|\alpha|}}{\partial x^\alpha}$$

It is trivial to show that these operators are linear functionals on the polynomial space. Now for $y \in \mathbb{C}^N$ and $g \in \mathbb{C}[x_1, \ldots, x_N]$ the differential functional $\partial_\alpha[y]$ is defined as

$$\partial_\alpha[y](g) = (\partial_\alpha g)(y)$$

This says that if we first evaluate the operator at a point and then apply it to a polynomial, this is the same as first applying the operator to the polynomial, then evaluating at the point. If we consider the vector space of all such differential operators, we can build the dual space.

DEFINITION 2.3.3. *Given a polynomial ideal $\mathcal{I} \subset \mathbb{C}[x_1, \ldots, x_n]$ the* ***Macaulay Dual Space*** *at the point $\hat{z} \in \mathbb{C}^n$ is the set of all differential functionals evaluated at $\hat{z}$ that vanish on $\mathcal{I}$, namely*

$$D_{\hat{z}}(\mathcal{I}) = \{\partial \in D_{\hat{z}} | \partial(g) = 0 \text{ for all } g \in \mathcal{I}\}.$$

Hauenstein showed in [13] that this definition can be extended to the homogeneous case for a homogeneous ideal $\mathcal{I}^h \subset \mathbb{C}[x_0, \ldots, x_N]$. We then get a homogeneous Macaulay dual space $D_y(\mathcal{I}^h)$.

DEFINITION 2.3.4. *Given a system of polynomial equations $\mathcal{F} = \{f_1, \ldots, f_n\}$ and a solution $\hat{z}$ to the system, the* ***Macaulay Matrix****, M, is a matrix built in degrees with the columns defined by the differential operators of each degree in each variable and the rows defined by the functions raised to the appropriate degree by each monomial and evaluated at the point $\hat{z}$.*

This definition is best followed by an example. Consider the homogeneous system of equations:

$$f_1 = x^2 + 6xy + 4xz + 9y^2 + 12yz + 4z^2 - 9w^2$$

$$f_2 = 4x^2 + 10xy + 9xz - 6y^2 - yz + 2z^2 - w^2$$

$$f_3 = 5x^2 + 12xy + 9xz - 15x - 9yz - 9y^2 + 9wy - 2z^2 + 3wz - w^2$$

We can build the degree 2 section of the Macualay matrix at the point $\hat{z} = \bar{0}$.

$$\begin{array}{c} \\ f_1 \\ f_2 \\ f_3 \end{array} \begin{pmatrix} \partial_{w^2} & \partial_{wx} & \partial_{wy} & \partial_{wz} & \partial_{x^2} & \partial_{xy} & \partial_{xz} & \partial_{y^2} & \partial_{yz} & \partial_{z^2} \\ \hline -9 & 0 & 0 & 0 & 1 & 6 & 4 & 9 & 12 & 4 \\ -1 & 0 & 0 & 0 & 4 & 10 & 9 & -6 & -1 & 2 \\ -1 & -15 & 9 & 3 & 5 & 12 & 9 & -9 & -9 & -2 \end{pmatrix}$$

In the standard definition of the Macaulay matrix, all lower degrees would be included. Since this system is homogeneous of degree 2, the degree 0 and degree 1 sections of the matrix would be rows and columns of zeros. This means that the null space of the degree two slice is the same as the whole Macaulay matrix up to degree two. We will use this fact and the following theorem to build our first algorithm.

THEOREM 2.3.5. *The null space of the Macaulay matrix forms a basis for the Macaulay dual space.*

Proof: *Given definition 2.3.3, we need* $\partial \in D_y | \partial(g) = 0$ *for all* $g \in \mathcal{I}$. *By definition 2.3.4, this is exactly the null space of the Macaulay matrix.*

□

2.3.1. DUAL OPERATIONS. In order to operate on the dual basis, we need to define some tools. Stetter and Thallinger in [22] and [23] , respectively, define a differential operator which we will need for preconditioning.

DEFINITION 2.3.6. *Given the dual space $D_y(\mathcal{F})$, with $\mathcal{F} \subset \mathbb{C}[x_1, \ldots, x_N]$, $y \in \mathbb{C}^N$, we have the* ***Stetter-Thallinger Operator*** *$\Phi_j : D_y \to D_y$ for $j = 1, \ldots, N$ with $\Phi_j(\partial_\alpha) = 0$ if $\alpha_j = 0$, and $\Phi_j(\partial_\alpha) = \partial_{\alpha - e_j}$ otherwise, where e_j is the standard basis vector.*

This operator has the effect of trimming degrees from the differential operators that form the basis of our vector space. It will be used in later algorithms to check for the "Closedness Condition". For now, we will use this operator in order to perform a *quotient ideal* in the dual space.

DEFINITION 2.3.7. *Let $\mathcal{I}$ and $\mathcal{J}$ be ideals in $\mathbb{C}[x_1, \ldots, x_n]$. The* ***Quotient Ideal*** *is defined $\mathcal{I} : \mathcal{J} = \{a \in \mathcal{J} | a\mathcal{J} \subset \mathcal{I}\}$.*

Given this definition, it follows that given $w \in \mathcal{I}$, we can create the quotient ideal $\mathcal{I} :< w >$. From classical algebraic geometry, we have that this process can be iterated, i.e., $< \mathcal{I} : w >: w = \mathcal{I} : w^2$. Since we have that $\mathcal{I} \subseteq \mathcal{I} : w \subset \mathcal{I} : w^2 \subseteq \ldots$ and there is an Ascending Chain Condition for Noetherian rings, we get that this must stabilize. This provides the following definition.

DEFINITION 2.3.8. *Let $\mathcal{I}$ and $\mathcal{J}$ be ideals in $\mathbb{C}[x_1 \ldots x_n]$. Consider the quotient ideal $\mathcal{I} : \mathcal{J}$ and the sequence $\mathcal{I} : \mathcal{J}^j$. There exists ℓ such that for $j = \ell + 1 \ldots \infty$, $\mathcal{I} : \mathcal{J}^j = \mathcal{I} : \mathcal{J}^\ell$. We call this ideal the* ***Saturation*** *of $\mathcal{I}$ by $\mathcal{J}$ and denote it $\mathcal{I} : \mathcal{J}^\infty$. The index ℓ is called the* ***Index of Regularity****.*

This definition extends to the case of $\mathcal{I} :< w >$. These quotient ideals, geometrically, correspond to the set subtraction of the variety $\mathcal{V}(w)$ from the variety $\mathcal{V}(\mathcal{I})$. It is a trivial exercise to show that this definition extends to the case where $\mathcal{I}$ and $\mathcal{J}$ are homogeneous ideals.

Recently, is has been shown in [7] that we can use this quotient operation in the dual space to act on the ideal. The Stetter-Thallinger operator acts on the dual space in the form of this quotient by removing one layer of the homogenizing variable. We use this operator to "trim" the null space. Since the homogenizing variable plays the role of ∞, we have that we are geometrically removing pieces of the variety $V(\mathcal{I})$ that live at ∞. If we can take this calculation to saturation, we will have removed all extraneous pieces living at ∞. In this way, we will remove all paths that diverge to ∞ in the homotopy continuation calculation.

2.3.2. Multiplicity Structure. While we may be able to find a solution to a polynomial system, finding the multiplicity of such a solution can be more difficult. This information can be useful in many different scenarios. Understanding the multiplicity structure of the set of solutions gives access to a number of invariants such as depth, breadth, and a dual basis. The multiplicity can also be used to get information about the dimension of a component in the case of positive dimensional solutions. Since a single positive dimensional component has an infinite number of points that are solutions, we can only sample the component. For instance, in Bertini, when solving for positive dimensional solutions, a number of points are found by intersecting higher dimensional components with generic linear spaces. The multiplicity can give information about the dimension of the component on which the point sits. This is known as the Local Dimension Test [1]. This test allows for a user to determine all of the positive dimensional components of a given system. One effective method for finding this information uses the Macaulay dual space. In order to use this information, we will need to have a clear understanding of multiplicity. We will use a definition from Stetter [22] for the multiplicity.

DEFINITION 2.3.9. *The* ***Multiplicity*** *of an ideal* $\mathcal{I} \subset \mathbb{C}[\bar{x}]$ *at an isolated zero,* $\hat{z}$*, is* m *if the dimension of the dual space* $\mathcal{D}_{\hat{z}}(\mathcal{I})$ *is* m*.*

With this definition, we are able to find the multiplicity of a given solution, $\hat{z}$ to a system of polynomials $\mathcal{F}(\bar{x})$. From Theorem 2.3.5 we have that this multiplicity is equivalent to the dimension of the null space of the Macaulay matrix evaluated at the point $\hat{z}$. In 2005, Dayton and Zeng in [11] developed a new algorithm for this process. This numerical computation made finding the multiplicity a reasonable computation given a system $\mathcal{F}$ and a solution $\hat{z}$. The algorithm takes as input the system of polynomials, the set of variables, and the solution. It outputs the multiplicity, the depth, the breadth, and the dual space. It is this algorithm that is used for the Local Dimension Test in *Bertini*.

The algorithm takes the system of equations and builds up the Macaulay matrix starting in degree 0. From this base case, the Macaulay matrix is built up at each degree and a null space computation done to create the dual basis. At this point, a *closedness condition* is checked and the algorithm continues until the correct stopping criteria is met. In [11], the authors took advantage of the block structure of matrices used in the calculation to create a reasonable timing for benchmark systems. With this in mind, and the definition of the Macaulay matrix from above, it is easy to see that the creation of the Macaulay matrix is a major hurdle in the calculation. In the next section, a better approach is introduced.

2.3.3. CLOSEDNESS SUBSPACE. The approach to finding the multiplicity given in section 2.3.2 is inhibited by the sizes of Macaulay matrices. A good example of this is to see the number of rows and columns that are necessary to build these special matrices. Given a system of n equations in N variables, the number of rows in the complete Macaulay matrix of degree α is $\binom{\alpha-1+n}{\alpha-1}N$. Likewise the number of columns is $\binom{\alpha+n}{\alpha}$. Since these row and

column sizes are choose functions, they grow combinatorially. It is clear to see that as the degree, number of equations, number of variables grow, the size of the Macaulay matrix quickly becomes very large. These matrices become overwhelmingly large for reasonable computation. Even in the case of parallel architecture, this is not a tenable model for large systems. In 2009, Zeng improved upon the multiplicity idea in [25] with the concept of the *closedness subspace*. From [13] we have the following Lemma.

LEMMA 2.3.10. *Let* $\mathcal{I} =< f_1, \ldots, f_n >$ *be an ideal in* $\mathbb{C}[x_1, \ldots, x_N]$, $y \in \mathbb{C}^N$ *and* $\partial \in D_y$. *Then* $\partial \in D_y(\mathcal{I})$ *if and only if* $\partial(f_i) = 0$ *for* $i = 1 \ldots n$ *and* $\Phi_j(\partial) \in D_y(\mathcal{I})$ *for* $j = 1 \ldots N$.

This so-called closedness condition picks out only particular monomials that contribute nontrivially to the Macaulay dual space. Zeng capitalized on this condition and proved that the resulting subspace will yield the equivalent Macaulay dual space. So, instead of build the Macaulay matrix $M^{\alpha}_{\hat{z}}(\mathcal{I})$ of degree α only the closedness subspace $\mathcal{C}^{\alpha}_{\hat{z}}(\mathcal{I})$ is necessary.

DEFINITION 2.3.11. *Let* $\mathcal{I} =< f_1, \ldots, f_n >$ *be an ideal in* $\mathbb{C}[x_1, \ldots, x_N]$. *Let* c *denote a differential operator with* $c \in D_{\hat{z}}(\mathcal{I})$ *and* $\alpha \in \mathbb{N}$. *Then the* ***Closedness Subspace*** *of* $\mathcal{I}$ *at* $\hat{z}$ *of degree* α *is* $\mathcal{C}_{\hat{z}}{}^{\alpha}(\mathcal{I}) = \{c = \Sigma_{|j| \leq \alpha} c_j \partial_j[\hat{z}] \mid \Phi_\sigma(c) \in D_{\hat{z}}{}^{\alpha-1}(\mathcal{I}),\ \sigma = 1, \ldots, n\}$

Given $c_1, \ldots, c_m$, a basis for $\mathcal{C}^{\alpha}_{\hat{z}}(\mathcal{I})$, we can build a smaller matrix W_α with

$$W_\alpha = \begin{pmatrix} c_1(f_1) & c_2(f_1) & \cdots & c_m(f_1) \\ c_1(f_2) & c_2(f_2) & \cdots & c_m(f_2) \\ \vdots & \vdots & \ddots & \vdots \\ c_1(f_n) & c_2(f_n) & \cdots & c_m(f_n) \end{pmatrix}$$

It is clear from the construction of W_α that the size of the matrix has been greatly reduced. Now, for a system of n equations in N variables at a point $\hat{z}$ with multiplicity m,

we have a matrix of greatest size n x $(m+n)$. Zeng showed this in [25] with an example by which a 218,790 x 48620 Macaulay matrix is replaced with a 9 x 265 matrix from the closedness subspace.

With W_α in hand, it is now only necessary to find a basis for the null space of W_α. Zeng showed that this is isomorphic to the null space of the Macaulay matrix. Thus, through a much smaller matrix, we are able to both determine multiplicity and gain access to a basis for the Macaulay dual space.

In 2011, Hao, Sommese, and Zeng, in [12] took this computation one step further. Using an equation by equation method for the closedness subspace, they were able to again speed up the computation of the multiplicity structure. By taking further advantage of the structure of the matrices being used, the authors were able to vastly improve the timings on benchmark systems as compared with the original closedness subspace software. It is this more advanced work that we have incorporated into an H-basis algorithm.

Like with the Multiplicity Structure algorithm and the first Closedness Subspace algorithm, this is highly parallelizable. Since the algorithms are all designed to turn a nonlinear problem into one of linear algebra, the authors are able to take advantage of numerical linear algebra in order to parallelize. One complication that will come with using the closedness subspace is that the algorithm is not designed with h-bases in mind. Instead, the Macaulay dual space is produced as a byproduct in the quest for the multiplicity. This will cause some initial issues when using this method to create an H-basis. Fixing this link is relegated to the domain of future work.

CHAPTER 3

Preconditioning

Our goal in preconditioning is to remove the paths that lead to ∞ in order to smooth out the tracking for homotopy continuation. We start with a zero-dimensional polynomial system and homogenize it using a homogenizing variable w. This is accomplished by bringing every monomial up to the same degree by filling in with the appropriate degree of w. Example: $x^2 - 1 \rightarrow x^2 - w^2$. As mentioned above, homogenizing the system moves the variety into projective space with the homogenizing variable playing the role of ∞. From the homogenized system, we can build the Macaulay matrix at the point $\hat{z} = \bar{0}$. The point $\hat{z} = \bar{0}$ is not actually a point in projective space. However, this point encodes the multiplicity for every zero-dimensional point. The null space of this Macaulay matrix, or the closeness subspace matrix, will provide a basis for the Macaulay dual space at this point. As we showed in section 2.3.1, we can quotient out by the homogenizing variable using the Stetter-Thallinger operator. We will begin with a naive algorithm using the Macaulay matrix that was used as a proof of concept.

3.1. Naive Algorithm

Algorithm 1 takes in the homogeneous system of equations as the initial basis H_0 for the variety $V(F)$. This basis is built up to the next degree in order to check the dimension of the Macaulay dual space. This process is continued until the dimension of the degree d dual space matches that of the degree $d-1$ dual space. Once this is achieved, the resulting matrix is then trimmed using the Stetter-Thallinger operator. This trimming operation is the equivalent of the quotient operation. After each run through the main loop, a new basis of homogeneous polynomials is produced. When this basis has stabilized, then we have created

Algorithm 1 Naive Algorithm

Input: A system of polynomials, $F = \{f_1, \ldots, f_s\} \subset \mathbb{C}[x_1, \ldots, x_n]$
Output: A preconditioned homogeneous system $\{h_1, \ldots, h_r\} \subset \mathbb{C}[w, x_1, \ldots, x_n]$ satisfying the H-basis conditions

```
Homogenize system: F → F^h
H_0 = {F^h} this is our initial working set
Degree d = max_{j=1..s}(degree(f_j))
Initialize Macaulay matrix in degree d, M_d(H_0)
i = 1
while dim(H_i) ≠ dim(H_{i-1}) do
  while dim(Null(M_k(H_i))) ≠ dim(Null(M_{k+1}(H_i))) do
    Build Macaulay Matrix M_d(H_i) in degree d
    Find a Basis N̂ for the null space Null(M_d(H_i))
    d → d + 1
  end while
  Apply Stetter-Thallinger operator N̂ ⟶ N̂_trim
  Find Null(N̂_trim^T)
  i → i + 1
  Return Polynomials in degree d − 1 ⟶ H_i
end while
Return H_i
```

an H-basis and removed infinite paths from the homotopy continuation run. The dimension of the null space stabilizing corresponds to the saturation of the ideal, i.e., $< \mathcal{I} : w^\infty >$ when quotienting by the homogenizing variable w. There are many computational obstacles with this approach.

There is a lot of wasted computation, first, when we build the Macaulay matrix to the top degree of all polynomials. This leads to a great computational cost in building the Macaulay matrix since all monomials must be brought to the highest degree of the system F in every possible way. It may be that we are forced to introduce a much larger number of polynomials into our system as we try to build a lower degree polynomial up to a higher degree. In building the Macaulay matrix, each of these lower degree homogeneous polynomials must be multiplied by every monomial of appropriate degree in every variable. As the number of equations and the number of variables grows, this quickly becomes intractable.

The second issue is that by trimming down by only one degree using the Stetter-Thallinger operator, we are recomputing the null space far more often than necessary. If, at each step of the computation we are able to trim down in as many degrees as possible, then we are computing the null space of the Macaulay matrix far less often. We would like to use the dual basis as effectively as possible without having to do the laborious calculation unnecessarily. What we are really looking for are new polynomial relations between the generators of the system. If we can search down through all previous degrees at each step, we can add these generators back to the system. In this way, we are recomputing the null space less frequently as well as removing possibly more layers at each step.

In general, the size of the Macaulay matrices will always present a bottleneck for the computation. This is somewhat mitigated by the fact that we are not building the full Macauala matrix, but rather only the homogeneous degree d slice. As was discussed in Chapter 1, we will still get the correct basis for the dual space. Despite this, the sizes of the Macaulay matrix will, in general, still cause many unnecessary calculations. In our next version of the algorithm, some of this concern will be eliminated using an equation by equation method that looks at each degree individually. We will also trim down the matrices by applying the Stetter-Thallinger operator in every degree necessary in each step. This allows us to get as much information as possible from the dual basis before recomputing.

3.2. Advanced Algorithm

The next version of the algorithm, labeled Algorithm 2 has removed some of the inefficiency and redundancy from the first naive algorithm. However, this has made the algorithm much more complex. The algorithm employs an equation by equation and degree by degree approach. At each step in the outer loop, we are able to add in any new polynomials as we

Algorithm 2 More Advanced Algorithm

Input: Zero-dimensional system of polynomials $F = \{f_1, \ldots, f_s\} \subset \mathbb{C}[x_1, \ldots, x_n]$
Output: Homogeneous system $H = \{h_1, \ldots, h_r\} \subset \mathbb{C}[w, x_1, \ldots, x_n]$ satisfying H-basis conditions

```
Homogenize System F → F^h = {f_1^h, ..., f_s^h}
Find degrees of {f_1^h, ..., f_s^h}
d := minimal degree*
P := {f_i^h | deg(f_i^h) ≤ d}
M_d(P) := Macaulay matrix of degree d for P
N_d(P) := Null space of M_d(P)
Record dim(N_d(P))
d → d + 1
while dim(N_d(P)) ≠ dim(N_{d+1}(P)) do
  while Not DONE do
    Compute N_d = Null(M_d(P))
    for j = 1 ... d − 1 do
      Apply j^th Stetter-Thallinger Operator N_d → N_dj
      Update dim(N_dj)
      if dim(N_dj) lowers then
        Get coefficients of new polynomial relations
        P → P+ new polynomials
      end if
    end for
    Compute M_d(P)
    if dim_{j=1...d−1}(N_dj) doesn't decrease then
      d → d + 1
      DONE
    end if
  end while
  P → P + {f_i^h | deg(f_i^h) = d}
  Compute M_d(P)
  Compute N_d
end while
H = Null(Transpose(N_d))
RETURN H
```

* Minimal means the lowest degree such that degree ≥ 2 with ≥ 2 polynomials

move up in degree. Also, at each step in the inner loop, we are able to add in any polynomial relations that appear and update our working set.

The breakdown of this algorithm is a little more complicated. Like before, we start with a zero-dimensional system, $\mathcal{F}$, and homogenize the system using a homogenizing variable, w, i.e. $\mathcal{F} \to \mathcal{F}^h$. Previously, we would then start in the highest degree of the system. Instead,

we will start in the lowest degree possible. The starting point will be the lowest degree such that we have at least two polynomials and we have reached at least degree two. This will be our working set P which will be updated at each step. If the degree d is ceases to produce new relations and needs to be raised, i.e. $d \to d+1$, we will then include any polynomials of degree $d+1$ from the original set $\mathcal{F}^h$. In this way, the set is always updated to include all polynomials of the working degree.

Using the working set P, we will record the dimension of the null space in all available degrees. This will be used for comparison and is updated each time through the loop. We then move up one degree and start the algorithm again. Every time we move up a degree, we check our original set and add to our working set any polynomials of the new degree as described above.

At each step in the core of the algorithm, we are using the Stetter-Thallinger to trim down the null space matrix. This null basis, in the form of a matrix, is really the dual basis for the system. If, after trimming, the rank of this matrix drops, this corresponds to a new polynomial relation. We take any new polynomial relations, p_i, and add them into our working set P and check to see if they are linearly independent in the monomials. Adding in the new polynomial relations, we are decreasing the number of possible solutions to the system. By checking for linear independence in the monomial space, we are ensuring that efforts are not being duplicated.

At this point, given our updated system, we recompute our null space and continue to trim. When the rank in each degree stabilizes, we move outside the loop and start again. If the rank of the degree ℓ matrix matches the rank of the degree $\ell+1$ matrix, we say the computation has stabilized. We can then rebuild our working set of polynomials, P, from

the degree ℓ matrix of coefficients. This will produce a set of homogeneous equations in degree ℓ and we have our H-basis.

There is a small caveat that must be taken into account. In order to reach a true H-basis by the definition, we must still reach the largest degree of the system. It will, in certain cases, be more conducive to instead truncate the computation and build the current working set. At this point we can add in all higher degree terms which have not been incorporated, i.e. any terms of degree higher than the max degree attained. This will still produce a preconditioned system though it may not have removed all infinite paths from the homotopy continuation calculation. However, if the degree is large enough and the number of variables is large enough, it can be better to leave these terms and truncate rather than attempt to build such a large Macaulay matrix.

3.3. Using Closedness Subspace

For the next algorithm, we will use a naive approach to taking the closedness subspace into account. Using the *Multiplicity* algorithm found in [12], we will avoid the Macaulay matrix altogether and instead compute the dual space using the closedness subspace. The benefit is the removal of the largest bottleneck in the computation. Without the Macaulay matrix calculation, the algorithm should run faster. The trade off, however, is that we must return to a naive approach.

Since the *Multiplicity* algorithm is not intended for use in the calculation of an H-basis, it is not optimized for this purpose. The resulting dual space is actually a byproduct of the computation. As such, it is necessary to change the stopping criterion within the *Multiplicity* algorithm to look for the dimension of the null space to stabilize. With that change made, we have treated the *Multiplicity* algorithm as a sub-function to be called by the main algorithm.

Algorithm 3 Naive Closedness Subspace

Input: Zero-dimensional system of polynomials $F = \{f_1, \ldots, f_s\}$
Output: Homogeneous system $H = \{h_1, \ldots, h_r\} \subset \mathbb{C}[w, x_1, \ldots, x_n]$
satisfying H-basis conditions

Homogenize System $F \to F^h = \{f_1^h, \ldots, f_s^h\}$
$H_0 = \{F^h\}$**, this is our initial set**
Set *Zero* to an $n+1$ **vector of zeros**
Run *Multiplicity* using $\{x_1, \ldots, x_n\}$**, *Zero*, and** H_0
  α **= degree of stabilization in *Multiplicity***
  m **= stabilized multiplicity**
  h **= vector of Hilbert numbers**
  D_α **= matrix of dual basis vectors**
  $D_\alpha^T =$ **Transpose**(D_α)
Apply Stetter-Thallinger operator, $D_\alpha^T \to \hat{D}_\alpha$
while Not DONE do
  $N_\alpha =$ **Null**$(\hat{D}_\alpha)$
  $\hat{N}_\alpha =$ **Transpose**(N_α)
  $M =$**Monomials in degree** $\alpha - 1$
  $P = \hat{N}_\alpha * M$ **the set of homogeneous polynomials**
  Run *Multiplicity* on P
  if dim(D_α) **stabilizes then**
    DONE
  end if
  $D_\alpha \to \hat{D}_\alpha$
end while
Return P

As in the naive algorithm, we compute the dual space and concatenate it into a Matrix. We can then trim the matrix using the Stetter-Thallinger operator and get back to polynomials. One benefit to using the *Multiplicity* algorithm is that it provides the Hilbert series for the computation. This means that unlike in previous versions of the algorithm, we do not have to check for the dimension of the dual space or the dimension of any of the lower degree spaces. The degree of stabilization as well as all lower degree dimensions are available. These are already calculated within *Multiplicity* and are given for free as part of the output. We can use these to compare at each step to look for a drop in multiplicity corresponding to new polynomial relations. Currently we are not able to incorporate these new polynomial

relations into the working set using the *Multiplicity* algorithm. This will be discussed in more detail in Chapter 5.

We start with our system of equations and homogenize them to get an initial basis. This starting set, along with a vector of the variables and a zero vector of appropriate size are used as input to the *Multiplicity* algorithm. From here, we get a degree of stabilization, α, and the associated dual space. After transposing and trimming the dual space, we apply a null space calculation. This gives a matrix of coefficients for the monomials in one degree lower. We can then take the dot product of this with the monomials in order to return a system of homogeneous polynomials.

We then feed this new system of polynomials back into *Multiplicity* and look for drops in the dimension, specifically drops in the multiplicity m. Once the dimension of the dual space stabilizes between successive runs of *Multiplicity*, we will be left with an H-basis. While this has removed the issue of using the Macaulay matrices and thus sped computation of the dual space, the algorithm still suffers from other problems associated with the naive approach. Namely, we are still only trimming by one degree and we are forced to the highest degree of the system. These issues are discussed in more detail in Chapter 5.

CHAPTER 4

Examples

4.1. A Simple Example

We start with a simple system of two equations in two unknowns. Consider the following system:

$f_1 = x^2 - 9$

$f_2 = xy + 3y - 1$

We homogenize the system using the homogenizing variable w.

$f_1^h = x^2 - 9w^2$

$f_2^h = xy + 3yw - w^2$

It is clear from the Bézout count that this system has 4 paths using a total degree homotopy and, when solved, only one solution. When we run the system through Algorithm 2, we end up with two homogeneous linear equations in the three variables.

$h_1 = -9.473684210526e-01w + 3.178947368421e-01x - 3.789473684210e-02y$

$h_2 = -5.263157894737e-02w - 3.789473684210e-02x + 9.978947368421e-01y$

This produces the unique solution and is easily solved. For the sake of consistency, we could use homotopy continuation, resulting in only one path tracked. It is not the case that we will always end up with linear polynomials, this is just an example to illustrate the algorithm. We have removed the 3 extraneous paths that would normally track to ∞ and have simplified the system for use in homotopy continuation. This problem is small enough that we could do this process symbolically with either H-bases or Gröbner bases. We use numerical methods here only as an illustration. Next, we look at a system that is larger and more complex.

4.2. Eco 8

Our next example system comes from the world of economics from [19]. This is a system of 8 equations in 8 variables. The system consists of a linear equation, a quadratic equation, and 6 cubic equations. This presents several interesting aspects. The first is that, using Gröbner bases in order to find the Hilbert Polynomial for this system requires a minimum of degree 12. This system stabilizes in degree 4 using an H-basis and can be computed using only a maximum of degree 6. The other interesting piece is that we can go intrinsic on the linear equation in order to remove a variable and an equation. This is a concept that works well with the *Bertini* package since the software is set up to take full advantage of sub-functions.

The system is as follows:

$$f_1 = (x_1 + x_1x_2 + x_2x_3 + x_3x_4 + x_4x_5 + x_5x_6 + x_6x_7)x_8 - 1$$

$$f_2 = (x_2 + x_1x_3 + x_2x_4 + x_3x_5 + x_4x_6 + x_5x_7)x_8 - 2$$

$$f_3 = (x_3 + x_1x_4 + x_2x_5 + x_3x_6 + x_4x_7)x_8 - 3$$

$$f_4 = (x_4 + x_1x_5 + x_2x_6 + x_3x_7)x_8 - 4$$

$$f_5 = (x_5 + x_1x_6 + x_2x_7)x_8 - 5$$

$$f_6 = (x_6 + x_1x_7)x_8 - 6$$

$$f_7 = x_7x_8 - 7$$

$$f_8 = x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + 1$$

As stated above, this system stabilizes in degree 4, yet it is generated in degree 2. We are able to get a generating set of 18 homogeneous equations of degree 2. This leads to $2^8 = 256$

paths. This is a large savings over the 1458 paths traced out by a total degree homotopy. In the affine case, a total degree homotopy yields 1394 paths going to ∞.

One benefit of this algorithm is that we are able to specify the number of iterations for a given run. This is desirable since we may not need to go fully to saturation. In this example, we set the number of iterations to be 3. This is the number of times the inner loop stabilized and we had to move up in the outer loop. After three iterations, the system is in degree 4 and it's Hilbert function is $\{8, 27, 53, 72\}$. This says that there are 72 or fewer solutions. Since we have reached the maximum degree of the system, we know that we have not left anything out. Now, this system is generated in degree 2 and consists of 18 homogeneous polynomials. Though the algorithm has not completed and the quotient has not reached saturation, we have achieved all of the solutions.

This example illustrates the value of Algorithm 2. After only a few iterations, we were able to remove almost everything at ∞ and retrieve a (super)set of solutions. Table 4.1 shows what has been removed and the associated timings. The algorithm was coded in Matlab and created the pre-conditioned system in $19.507s$ on one processor. When taking the pre-conditioning time into account, this is not a great savings for the affine case, and is only moderate in the homogeneous case. The savings will be increased after taking advantage of parallelism and multi-core technology.

A separate concern is the issue of paths crossing during homotopy continuation. In both the affine and homogenous cases, there were an average of 11 possible path crossings per total degree run on the original system. This implies that the paths are not smooth and cause some problems numerically. After preconditioning, there were no path crossings in either the affine or the homogeneous case. This may be an interesting extra benefit of preconditioning using h-bases.

TABLE 4.1. Eco 8: precondition time- 19.507s

Affine			
	Total Paths	Infinite Paths	Total Degree Time
Original System	1458	1394	13.17s
Preconditioned System	256	16	7.80s
Homogeneous			
	Total Paths	Infinite Paths	Total Degree Time
Original System	1458	7	48.47s
Preconditioned System	256	0	9.65s

4.3. STEWART-GOUGH PLATFORM

We now take a look at a system of equations coming from kinematics. This particular system comes from a Stewart-Gough platform and can be found in [24]. This system has 9 equations in 9 variables with inexact coefficients. This means that symbolically computing the solutions to this system is difficult at best. Also, when run with a total degree homotopy, this system requires the tracking of 4096 paths. This is not insurmountable, but the system only has 40 solutions all of which are real. This means that the other 4056 paths are infinite and will cause computational problems when tracking paths.

After preconditioning, we arrive at a system of degree 2 homogeneous polynomials. Since there are 9 variables, this says that the homogeneous bézout count for the system is $2^9 = 512$. So, we only track 512 paths, all of which are nonsingular and none of which track to ∞. 40 of these are solutions and the rest are false solutions known as Bertini junk points. Moreover, the H-basis for this system stabilizes very quickly, within only a few iterations of the Algorithm 2. Computationally this is very fast and tracking the 512 paths is much more efficient than tracking the 4096 from the original system.

The algorithm runs the initialized homogeneous system in 78.50s in Matlab on one processor. A portion of that time is just the initial setup to find the degrees of the polynomials

and build the initial Macaulay matrix. The end result of the algorithm is a Bertini input file that we can then run using a total degree homotopy. Using the same processor, Bertini solved the preconditioned system in 25.33s. This gives us a total computational time of 103.83s. Comparatively, using a total degree homotopy on the original system using Bertini requires over 30 minutes.

Another idea was to see how the H-basis computation compared to method known as *regeneration*. Regeneration is a tool from numerical algebraic geometry that can reduce computation time and number of paths, see [14]. Running the original system in Bertini using regeneration on one processor yields varying times. When the regeneration run goes well, i.e., no major problems in the tracking, this is fast. However, it is possible for the regeneration run to get hung up on the original system. In testing, this happened about a third of the time. With the preconditioned system, the regeneration run was consistently smooth and without issues. For all of the details, see Table 4.2.

TABLE 4.2. Stewart-Gough Platform System: precondition time- 78.50s

	Original System	Preconditioned System
Total Degree	30m 14.70s	25.33s
Paths	4096	512
Infinite Paths	4056	0
Regeneration	106.32s	70s
Paths	160	512
Infinite Paths	120	0

4.4. Reimer Sphere System

The next system is used both as a proof of concept of the Closedness Subspace algorithm as well as as an illustration of its limitations. This is a system of equations, based on spherical equations presented in [20]. Algorithm 1 was used on this system in [15] to find an H-basis

and precondition the system. For this example, we used both Algorithm 2 and the naive closedness subspace method of Algorithm 3. The system is as follows:

$$f_1 = -1 - 2x^2 - 2y^2 - 2z^2$$

$$f_2 = -1 - 2x^3 - 2y^3 - 2z^3$$

$$f_3 = -1 - 2x^4 - 2y^4 - 2z^4$$

This is a simple system of three equations in three variables. The original system tracks 24 paths using a total degree homotopy. The system only has 12 solutions and the other 12 paths track to ∞. After computing an H-basis, the system tracks 27 paths using a total degree homotopy. This tracks the 12 solutions along with 15 finite non solutions. While the system creates more possible solutions, it fully removes any infinite paths. This higher number of solutions is a symptom of the fact that the system stabilized in degree 3 and has 3 variables. This creates $3^3 = 27$ paths. We were able to use the closedness subspace in order to avoid the Macaulay matrices and find the preconditioned system. This was novel and worked as a nice proof of concept but there were slight problems.

When computed using Algorithm 2, the system stabilizes very quickly. However, using the closedness subspace with Algorithm 3, the system takes much longer. See Table 4.3 for timings. While this is just one example, it is representative of the phenomena that this naive algorithm is not as fast or efficient as Algorithm 2.

TABLE 4.3. Reimer 3 Timings

	Algorithm 2	Algorithm 3 (Closedness Subspace)
Preconditioning	5.8s	54.5s
Paths	27	27
Solutions	12	12
Infinite Paths	0	0

The discrepancy here can be attributed to several factors. The first is that the closedness subspace algorithm was never designed to find an H-basis. This will be discussed more thoroughly in Chapter 4. The second and most important factor is that, since the *Multiplicity* algorithm is designed to stand on its own, there is a lot of redundant work being done.

A prime example of this is the conversion from polynomials to matrices and back. In Algorithm 2, once the polynomials have been converted to matrices, the coefficients live as matrices until the very end of the algorithm. This was optimized within the algorithm to be cost effective. Since the transition from symbolic polynomials to numeric matrices is costly and cannot be parallelized, this is computationally expensive. In Algorithm 3, the polynomials are converted by *Multiplicity*, then must be converted back to polynomials in order to feed back into the engine. This was one of the main problems with the naive approach that Algorithm 2 attempted to fix. Unfortunately, without completely changing the *Multiplicity* algorithm, there is no immediate fix to this bottleneck. This is further discussed in Chapter 4.

In general, it is known that the closedness subspace can find a Macaulay dual space much faster than finding the null space of a Macaulay matrix. The algorithm, though, relies on pieces that are not easily broken apart and changed. And, as shown above, using the closedness subspace algorithm naively as an engine is not efficient. Thoughts for fixing this are presented in the next chapter.

CHAPTER 5

Future Work

5.1. Equation by Equation

The next step is to write more efficient code for the algorithm. While the *Multiplicity* algorithm computes the dual space much faster, the code is not efficient for H-basis calculations. Instead, the algorithm has been used naively as an engine for producing fast dual bases. A better step would be to rewrite the algorithm to optimize specifically for the H-basis calculation.

As we saw in section 2.3.3, employing the closedness subspace allows for the use of much smaller matrices. Finding a basis for the Macaulay dual space is reduced to finding the null space of the W_α matrices as opposed to the null space of the Macaulay matrices. Using the equation by equation method of the closedness subspace from [12], we are able to severely reduce the amount of computation needed to obtain a basis for the dual space. However, these computation savings are more than offset by the transition costs of using *Multipliticy*. There are a few questions that come along with optimizing this in order to compute an H-basis. This work will necessarily have to take place inside the *Multiplicity* algorithm while still satisfying the conditions of the closedness subspace.

The main challenge is that the closedness subspace software is not designed to look for the same information as the H-basis software. The calculation finds a basis for the dual space, but does not return to the original polynomial space. Initially, the best course of action was to get the closedness subspace to return homogeneous polynomials at each step. It was necessary to adapt the correct stopping criteria necessary to find an H-basis. This allowed for the use of the code as an engine but was very inefficient as a means to find an

H-basis. We would like to take advantage of the computation savings inherent in Algorithm 2. The *Multiplicity* code does not allow for the possibility of adding in polynomials of the top degree during a loop. In this way, we are inhibited from starting in the lowest degree possible and trimming on subsets of the polynomial ideal. New work needs to be done to investigate how to implement these methods within the *Multiplicity* algorithm or whether that is possible without destroying the algorithm.

This poses several new and interesting questions. The first question is how to adapt the Stetter-Thallinger operator in order to quotient out by the homogenizing variable effectively. While we have shown that we can apply the Stetter-Thallinger operator in order to trim the dual space, we are currently unable to add in new polynomial relations found from consecutive trimming. We would like to be able to add in the new polynomial relations found and show that we do not need to recompute the closedness subspace. This would require that the new polynomial relations that come from trimming satisfy the closedness condition.

The second question revolves around whether we can iterate as in Algorithm 2 once we have computed the basis for the Macaulay dual space. This is very closely related to the previous question. We would like to create an inner loop that trims as much as possible before starting a new closedness subspace calculation. In this way, it would be possible to save even more computation time and take advantage of lower degree polynomials. Little is known as to whether these ideas can actually be implemented into a closedness subspace algorithm.

With these questions answered, there is also the option of taking better advantage of parallelism and multi-core technology. Since the Closedness Subspace algorithm uses numerical linear algebra, parallelizing will be a very reasonable step. This will create speed ups, especially in the case of large systems where the size of the W_α matrix grows large.

5.2. Positive Dimensional Systems

Within this monograph, we have been focusing on polynomial systems with zero-dimensional solution sets. However, it is often the case that these systems have positive dimensional solution sets. These could be curves, surfaces, or higher dimensional objects. With the algorithms given, there is no stopping criterion for positive dimensional solutions. The current stopping criterion relies on the fact that the Hilbert polynomial is constant for zero-dimensional ideals. For positive dimensional ideals, the Hilbert polynomial is a polynomial of the degree matching the top dimension of the variety.

Recent work has allowed for the possibility of a new stopping criterion that would work for positive dimensional ideals. This is based on work done by Greg Reid of University of Western Ontario. Using the same Macaulay matrix approach, there will be a reasonable stopping criterion in order to find an H-basis. This goal, while interesting mathematically, is not really necessary as it may be possible to precondition these systems without going to saturation. In that case, we do not actually need an H-basis, but instead new polynomial relations from the H-basis computation.

BIBLIOGRAPHY

[1] D.J. Bates, J.D. Hauenstein, C. Peterson, and A.J. Sommese, *A numerical local dimensions test for points on the solution set of a system of polynomial equations*, SIAM J. Numer. Anal., 47, 2009, no. 5, pp. 3608–3623.

[2] D.J. Bates, J.D. Hauenstein, A.J. Sommese, and C.W. Wampler, *Bertini: software for numerical algebraic geometry*, Available at `www.nd.edu/∼sommese/bertini`.

[3] D.J Bates, J.D. Hauenstein, A.J. Sommese, and C.W. Wampler, *Stepsize control for path tracking*, Contemporary Mathematics 496:21-31, 2009.

[4] D.J. Bates, J.D. Hauenstein, A.J. Sommese, and C.W. Wampler, *Adaptive multiprecision path tracking*, SIAM J. Numer. Anal., 46, 2008, no. 2, pp. 722–746.

[5] D.J. Bates, C. Peterson, A.J. Sommese, *A numerical-symbolic algorithm for computing the multiplicity of a component of an algebraic set*, Journal of Complexity, 22 (4), pp. 475–489, 2006.

[6] D.J. Bates, J.D. Hauenstein, A.J. Sommese, and C.W. Wampler, *Numerically solving polynomial systems with bertini*, Software, Environments, and Tools 25, SIAM, 2013.

[7] D.J. Bates, J.D. Hauenstein, and S.L. Ihde *Preconditioning polynomial systems via h-bases*, preprint, 2013.

[8] D.A. Brake *Homotopy Continuation Methods, Intrinsic Localized Modes, and Cooperative Robotic Workspaces*, PhD Dissertation, 2012.

[9] D. Cox, J. Little, and D. O'Shea, *Using Algebraic Geometry*, Graduate Texts in Mathematics, Springer, New York, 1998.

[10] D. Cox, J. Little, and D. O'Shea, *Ideals, varieties, and algorithms*, third ed., Undergraduate Texts in Mathematics, Springer, New York, 2007.

[11] B.H. Dayton and Z. Zeng, *Computing the multiplicity structure in solving polynomial systems*, In Proceedings of ISSAC'05, (2005), pp. 116–123.

[12] W. Hao, A. J. Sommese, and Z. Zeng, *An algorithm and software for computing multiplicity structure at zeros of nonlinear systems*, *ACM Transactions of Mathematical Software*, 2012.

[13] J.D. Hauenstein, *Algebraic computations using Macaulay dual spaces*, preprint, 2011.

[14] J.D.Hauenstein, A.J. Sommese, and C.W. Wampler, *Regeneration homotopies for solving systems of polynomials*, Mathematics of Computation, 80, pp. 345-347, 2011.

[15] S.L. Ihde *Preconditioning polynomial systems for homotopy continuation* Master's Thesis, Colorado State University, 2011.

[16] T.L Lee, T.Y. Li, and C.H. Tsai, *Hom4ps-2.0, a software package for solving polynomial systems by the polyhedral homotopy continuation method*, Computing, 83, pp. 109-133, 2008.

[17] F.S. Macaulay, *The algebraic theory of modular systems*, Cambridge University Press, 1916.

[18] H.M. Möller, J. Sauer, *H-bases for polynomial interpolation and system solving* Advances in Computational Mathematics, v.12, 2000.

[19] A. Morgan, *Solving polynomial systems using continuation for engineering and scientific problems* Prentice Hall, New Jersey, 1987.

[20] M. Reimer *Constructive theory of multivariate functions*, BI Wissenschaftsverlag, Mannheim, 1990.

[21] A.J. Sommese and C.W. Wampler, *The numerical solution to systems of polynomials arising in engineering and science*, World Scientific, Singapore, 2005.

[22] H.J. Stetter, *Numerical polynomial algebra*, SIAM, 2004.

[23] G.H. Thallinger, *Analysis of zero clusters in multivariate polynomial systems*, Diploma Thesis, Tech. Univ. Vienna, 1996.

[24] J. Verschelde *PHCpack: a general-purpose solver for polynomial systems by homotopy continuation* ACM Transactions of Mathematical Software, 1999.

[25] Z. Zeng, *The closedness subspace method for computing the multiplicity structure of a polynomial system*, 2009.