DISSERTATION

AVOIDING SINGULARITIES DURING HOMOTOPY CONTINUATION

Submitted by

Timothy E. Hodges

Department of Mathematics

In partial fulfillment of the requirements

For the Degree of Doctor of Philosophy

Colorado State University

Fort Collins, Colorado

Spring 2017

Doctoral Committee:

Advisor: Daniel J. Bates

A. P. W. Böhm
Alexander Hulpke
Chris Peterson

ABSTRACT

AVOIDING SINGULARITIES DURING HOMOTOPY CONTINUATION

In numerical algebraic geometry, the goal is to find solutions to a polynomial system $F(x_1, x_2, ...x_n)$. This is done through a process called homotopy continuation. During this process, it is possible to encounter areas of ill-conditioning. These areas can cause failure of homotopy continuation or an increase in run time. In this thesis, we formalize where these areas of ill-conditioning can happen, and give a novel method for avoiding them. In addition, future work and possible improvements to the method are proposed. We also report on related developments in the Bertini software package. In addition, we discuss new infrastructure and heuristics for tuning configurations during homotopy continuation.

ACKNOWLEDGEMENTS

I would not be here without the help of my Ph.D. advisor, Dan Bates. Dan was infinitely patient and understanding throughout my tenure as his student, for which I am forever grateful. Many students may know Dan, and seek him for guidance, but I am one of the lucky that has been able to call him my advisor. Dan, thank you for all the opportunities and experiences you have given me.

In addition, I would like to thank my committee: A.P. Bohm, Alexander Hulpke, and Chris Peterson. All three of you have been instrumental in my education, undergraduate and graduate, and cannot be thanked enough. I appreciate all the challenges and discussions you have put before me. I truly have been lucky to have a committee filled with professors who helped shape who I am.

To my parents, Glen and Sue Hodges, thank you so much for support that is uncountable. You have assuredly been waiting for my departure from school! (Don't worry I am ready to leave.) I could write an entire essay on the ways you both have helped me. Just know I am the luckiest guy to have you both as my parents.

To my friends, thank you all for keeping me sane, being there at the hardest of times, and giving me some amazing memories. Our friendships are priceless, and I look forward to growing our friendships as we scatter across the world. All of you are more important to me than I can ever explain, especially in an acknowledgements section.

TABLE OF CONTENTS

Chapter 1

BACKGROUND

This section gives a historical context to the reader for why we consider a set of polynomials and the corresponding set of solutions. For this reason, if the reader does not wish to read Section 1.1 they can skip to Section 1.2 where we start to define the necessary objects.

1.1. CLASSICAL ALGEBRAIC GEOMETRY

Given a set of polynomial equations, can we find the set of common solutions? This question fueled the development of classical algebraic geometry. On one side, we have a set of algebraic objects called polynomials. On the other is the corresponding geometric object, the set of common solutions. Given a system of polynomials $\{f_1, f_2, ...f_n\}$ in variables $\{x_1, x_2, ..., x_n\}$, one may define the **ideal**, $I = \langle f_1, f_2, ..., f_n \rangle$, generated by $f_1, f_2, ...f_n$. An ideal is a fundamental object in the area of Abstract Algebra [13]. For our discussion we will consider a finite set of polynomials that generate an ideal.

For the purposes of this thesis, we choose to have our coefficients lie in the field of complex numbers, denoted $\mathbb{C}$. The field of complex numbers is algebraically closed. This ensures that all solutions to a polynomial with coefficients in $\mathbb{C}$ will also be in $\mathbb{C}$ [13]. Below is an example of why the real numbers, $\mathbb{R}$, are not algebraically closed.

Example 1.1.1. *The polynomial $f = x^2+1$ has no solutions over the field of real numbers $\mathbb{R}$. If we instead choose the algebraically closed field of complex numbers, $\mathbb{C}$, then f has two solutions $x = \pm i$.*

If we consider a finite subset of elements $f_1, f_2, ..., f_k$ in a polynomial ring $\mathbb{C}[z_1, z_2, ..., z_n]$, can we find the set of points $(z_1, z_2, ..., z_n)$ that satisfy all polynomials $f_1, f_2, ..., f_k$ simultaneously? These points may be isolated or may lie on a higher dimensional set of solutions, such as a curve or a surface. The object these points make is called a **variety**. To learn more about varieties, refer to [11, 13, 17, 18].

EXAMPLE 1.1.2. *Consider the set of bivariate polynomials with complex coefficients. The set of all such polynomials is denoted* $\mathbb{C}[x, y]$. *The ideal* $I =< y - x^2, y^2 - 1 >$ *consists of all polynomial combinations of* $y - x^2$ *and* $y^2 - 1$ *i.e.,* $f(x, y)(y - x^2) + g(x, y)(y^2 - 1)$ *for any* $f, g \in \mathbb{C}[x, y]$.

For the first generator, $y - x^2$, *the solution set is the complex parabola* $y - x^2 = 0$, *or* $y = x^2$. *For the generator* $y^2 - 1$, *we see that* $y = \pm 1$.

Since we seek to solve these two polynomials simultaneously, we must consider the intersection of their solution sets.

If $y = 1$, *then* $y = x^2$ *gives us* $x = \pm 1$. *For* $y = -1$, *we see that* $x = \pm i$.

Intersecting the solution sets gives us all possible solutions because we are working over the complex numbers. Our variety, or solution set, is thus the set of four points,
$\{(1, 1), (-1, 1), (i, -1), (i, -1)\}$. *This example illustrates why we must be careful not to visualize in* $\mathbb{R}$, *because we risk missing the solutions in complex space.*

The computation of varieties and ideals has a rich history but is still a very active area. How to compute these varieties when the ideals are nontrivial is the subject of the next section.

1.2. SYSTEMS OF POLYNOMIALS

Going back to the 1960's, there are algebraic methods for finding the common solutions of a polynomial system. We cover them briefly for context before moving to a modern geometric method, *homotopy continuation*. We reduce the notation of $z_1, z_2, ..., z_n$ to $\mathbf{z}$. Consider now a system of polynomials:

$$F(\mathbf{z}) = \{f_1(\mathbf{z}), f_2(\mathbf{z}), ..., f_k(\mathbf{z})\}, \quad \mathbf{z} \in \mathbb{C}^n$$

The system $F(\mathbf{z})$ has k equations in n unknowns. We can always take random linear combinations of the f_i to create a *square system*, thus making both the number of equations and the number of variables n. At most, this will add solutions that do not solve the original $f_1, f_2, ..., f_k$, but these points can be removed easily [8].

How would we go about finding solutions that make $f_1 = f_2 = \cdots = f_k = 0$? One method developed by Sylvester can be used to find the common roots to two univariate polynomials. Sylvester's use of *resultants* required the use of determinants of matrices that are filled with the coefficients of the polynomials [11, 16]. This method was extended to more than one variable but rapidly becomes more computationally expensive because of the determinant calculations needed [15].

Another idea of how to solve this problem comes from how we normally solve linear equations. Can we solve for one of the variables and back substitute to solve for the others? This may not be entirely possible given the original set of polynomials. The idea behind **Gröbner bases** is to make this method work.

Gröbner basis methods create a different set of polynomials $p_1, ..., p_m$ with the same solutions. This set has the property of having a unique remainder, given a particular ordering on the monomials. This method is the right tool for many applications in Algebraic Geometry [11–14, 16]. When the number of equations or variables grows too high, though, this method also becomes computationally expensive [26].

We move to a modern *geometric* method of computing solutions. This method requires that we design another polynomial system $G = \{g_1, g_2, ...g_k\}$ that has at least as many solutions as $F(\mathbf{z})$. We can then construct a homotopy to deform the solutions of $G(\mathbf{z})$ to the solutions of $F(\mathbf{z})$ with extraneous solutions going off to ∞. This method is known as **homtopy continuation**.

1.3. HOMOTOPY CONTINUATION

Homotopy continuation is a technique for approximating solutions of a system $\mathbf{F}(\mathbf{z}; \mathbf{p}) = 0$ of n general nonlinear equations in variables $\mathbf{z} \in \mathbb{C}^N$ and parameters $\mathbf{p} \in \mathbb{C}^k$. In this setting, known solutions of system $\mathbf{F}(\mathbf{z}; \mathbf{p_0}) = 0$ at an initial parameter value $\mathbf{p} = \mathbf{p_0}$ vary as $\mathbf{p}$ moves along parameter space path $\eta \subset \mathbb{C}^k$ from $\mathbf{p_0}$ to a final parameter value $\mathbf{p_1}$. The m solutions $\mathbf{z}_\mathbf{0}^{(\ell)}$, $\ell = 1, \ldots, m$, of $\mathbf{F}(\mathbf{z}; \mathbf{p_0}) = 0$ are the starting points of *solution paths* in $\mathbb{C}^N$ that we may follow to the solutions of $\mathbf{F}(\mathbf{z}; \mathbf{p_1}) = 0$. These m paths are followed approximately (*tracked*) via numerical methods, described in great detail in [1].

By specializing to the case of systems of *polynomial* equations with n equations and n variables, we can give names to a few members of this family of polynomials. The system we wish to solve, $\mathbf{F}(\mathbf{z}; \mathbf{p_1}) = 0$, we call the *target system.* The starting points of our solution paths, are the solutions the *start system*, $\mathbf{F}(\mathbf{z}; \mathbf{p_0}) = 0$

In basic homotopy continuation, $k = 1$, so we change our notation to $\mathbf{H}(\mathbf{z}; t)$, with $\mathbf{H}$ reflecting the fact that we now have a homotopy function.The canonical choice of homotopy function is the *straight-line homotopy* [2],

$$\mathbf{H}(\mathbf{z};t) = \mathbf{f}(\mathbf{z})(1-t) + t\mathbf{g}(\mathbf{z}) \tag{1.3.1}$$

This choice of homotopy glues together a *start system* $\mathbf{H}(\mathbf{z};1) = \mathbf{g}(\mathbf{z})$ with m known solutions and a *target system* $\mathbf{H}(\mathbf{z};0) = \mathbf{f}(\mathbf{z})$ that we wish to solve.

There are many well-known methods for choosing $\mathbf{g}(\mathbf{z})$ [8]. For example, we *could* do this by defining:

$$g_i(\mathbf{z}) := z_i^{deg(f_i)} - 1, \quad i = 1, ..., n$$

Then the polynomials in $\mathbf{f}(\mathbf{z})$ and $\mathbf{g}(\mathbf{z})$ have the same degree. The definition of $g_i(\mathbf{z})$ gives us the roots of unity around the circle in the complex plane defined by the complex variable z_i. The number of roots of unity is determined by the degree of $f_i(\mathbf{z})$. With this construction we have a *start system*, $\mathbf{g}(\mathbf{z})$, that has the maximum number of distinct solutions that $\mathbf{f}(\mathbf{z})$ could possibly have. This number is known as the total degree bound or *Bézout bound* [8]. This bound is one of many upper bounds, some of which may be lower. These various types of bounds yield different choices of $\mathbf{g}(\mathbf{z})$ that may be easier or harder to solve. For more options on *start systems* and their bounds, see [8].

The homotopy function

$$\mathbf{H}(\mathbf{z};t) = \mathbf{f}(\mathbf{z})(1-t) + t\mathbf{g}(\mathbf{z})$$

is defined by the variables $\mathbf{z} \in \mathbb{C}^N$ and $t \in \mathbb{C}$, the latter of which is called the *path variable*. The name *path variable* is appropriate because as we deform the solutions of $\mathbf{g}(\mathbf{z})$ into solutions of $\mathbf{f}(\mathbf{z})$ we will be carving a path in the complex plane $\mathbb{C}$. This path will be random so as to avoid issues that will be discussed in Section 1.4. At values of t away from 1 and 0, we have a combination of the solutions of F and G, and with probability one they will all be distinct.

Basic numerical continuation amounts to a combination of numerical linear algebra and some bookkeeping. The atomic procedure is the predictor-corrector step. Briefly, given a point $\mathbf{z}(t_1)$ on (or very near) a solution path where $t = t_1$, there is a two-phase procedure to compute an approximation $\mathbf{z}(t_2)$ to a point on the solution path, for $t_2 = t_1 - \Delta t$. A *prediction* (e.g., from t_1 to t_2 along the tangent direction from $\mathbf{z}(t_1)$, as in Euler's method) is followed by a *correction* phase, typically consisting of one or more steps of Newton's method with t frozen at t_2. A schematic of one step is given in Figure 1.1. Such computations involve only numerical linear algebra, i.e., solving a linear system $\mathbf{Jx} = \mathbf{y}$ for various vectors $\mathbf{x}, \mathbf{y}$ and the Jacobian matrix $\mathbf{J}$, consisting of all first partial derivatives of the polynomials of $\mathbf{H}(\mathbf{z}, t)$ with respect to variables $\mathbf{z}$. If all goes well, this process will continue to $t = 0.1$, at which time specialized algorithms, called endgames, are used to finish the deformation for reasons described in section 1.4

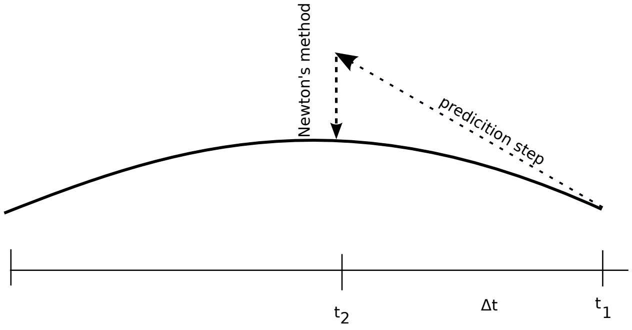

FIGURE 1.1. One step in the predictor-corrector scheme.

A *prediction* can be done multiple ways varying from an Euler prediction to higher order predictors that include an error estimate, e.g., Runge-Kutta-Fehlberg45 [23]. These methods are numerical ODE (ordinary differential equations) methods. A myriad of prediction methods can be used in the *Bertini* software package [8, 5]. All work in this dissertation uses *Bertini* [5], a numerical algebraic geometry software package, with MATLAB [25] and *BertiniLab* [9], a module to help use MATLAB and *Bertini* together. There are other software packages using homotopy continuation, see [10, 34].

What can go wrong during homotopy continuation? There may be numerical issues that cause predictions to be outside of the basin of convergence for Newton's method. These issues occur when two paths are near each other, or if they collide at a *singularity*. These collisions almost always occur at our target system if anywhere, but can occur outside of $t = 0$. In this thesis, we will be attempting to avoid these singularities. In the next section, we formalize what it means to be a singularity.

1.4. BRANCH POINTS AND RAMIFICATION POINTS

By virtue of the construction of $\mathbf{H}(\mathbf{z}; t)$, for almost all choices of $t \in \mathbb{C}$, the fiber of projection $\pi : \mathbb{C}^N \times \mathbb{C} \to \mathbb{C}$ onto the last component consists of the same number of points.The solution set in this case will have ℓ distinct solutions that all have multiplicity one[1]. However, there does exist a finite exceptional set of points in $\mathbb{C}$ where there is at least one solution of multiplicity greater than one. More precisely, the set $\mathcal{B} \subset \mathbb{C}$ consisting of points with fewer than ℓ solutions is the solution set of a polynomial system. $\mathcal{B}$ is a solution set for a polynomial system with $n + 1$ equations and n variables. This structure will make the dimension of $\mathcal{B}$ a lower dimension than the ambient space $\mathbb{C}$. Hence, $\dim(\mathcal{B}) = 0$ and there are only finitely many points in the t-plane at which the number of solutions of $\mathbf{H}(\mathbf{z}; t) = 0$ is less than ℓ. It is for this reason that a randomly-chosen path $\eta \subset \mathbb{C}$ will contain no points of $\mathcal{B}$ *with probability one.*

[1] More specifically, there is a Zariski open, dense subset of $\mathbb{C}$ for which this is true, the complement of which is a finite set of points.

Points $b \in \mathcal{B}$ are called **branch points**. The corresponding solution $\mathbf{z}^*$ of $H(\mathbf{z}, b)$ is called a **ramification point**. In the picture below, the **X** represents the location of a branch point in the plane defined by the path variable. Above the branch point is the ramification point denoted by a star. Notice that only the dashed path has a ramification point. The black path downstairs and corresponding solution curves upstairs do not encounter a singularity.

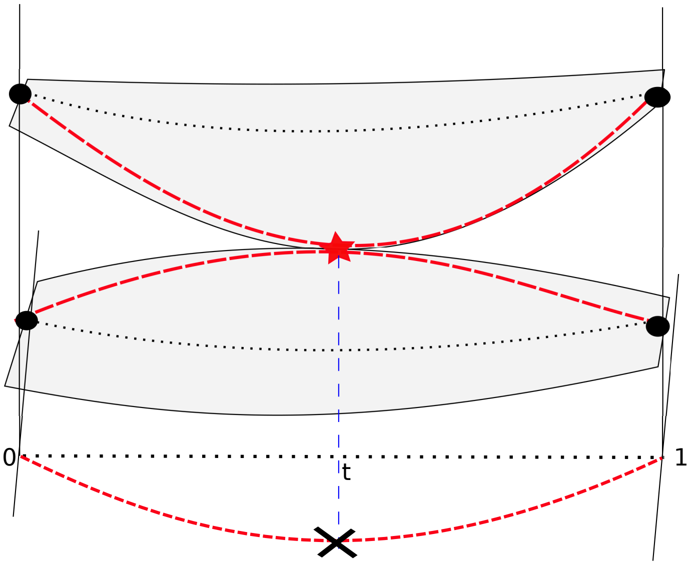

FIGURE 1.2. A 3-D depiction of homotopy continuation. The dotted path on the bottom represents a curve traveled where no branch points are encountered. The dashed path has a branch point at the X. The star above denotes the corresponding ramification point. The colors of the paths above are on the sheets created by homotopy continuation.

Upon choosing a random path from $t = 1$ to $t = 0$ in the complex plane, there is a probability one gurantee of avoiding these branch points while doing homotopy continuation. One method to achieve this randomness in η is the *gamma trick* [30], where a random $\gamma \in \mathbb{C}$ is multiplied into the second term of homotopy $\mathbf{H}(\mathbf{z}; t)$ in (1.3.1).

However, with numerical approximations, we are working in finite precision and can move close enough to one of these points to encounter numerical issues. This is vastly more likely to happen than encountering the actual branch point. These *ill-conditioned zones* around branch points cause multiple issues. Using adaptive precision can counter this numerical ill-conditioning and is described briefly below.

Given a matrix $\mathbf{A} \in \mathbb{C}^{n\times n}$ and vector $\mathbf{b} \in \mathbb{C}^n$, suppose we wish to approximate the solution $\mathbf{x} \in \mathbb{C}^n$ of $\mathbf{Ax} = \mathbf{b}$ so that the computed solution $\hat{\mathbf{x}}$ satisfies $|\mathbf{x} - \hat{\mathbf{x}}| < \delta$ for some specified tolerance $\delta \in \mathbb{R}^+$ for some choice of norm. Working with $\mathbf{P}$ digits of precision, a well-known result of Wilkinson [35] gives us that the number of accurate digits of $\hat{\mathbf{x}}$ is approximately

$$\mathbf{P} - \log(\kappa(\mathbf{A})),$$

where $\kappa(\mathbf{A})$ is the condition number of $\mathbf{A}$ [8, 21, 23]. Thus, to retain a desired accuracy during continuation, one may actively adapt precision $\mathbf{P}$ during each predictor-corrector step, based on approximations of the condition number of the *Jacobian* matrix, $\kappa(\mathbf{J})$ [6, 7].

Newton's method uses the *Jacobian* matrix, denoted $\mathbf{J}(\mathbf{z}; t)$. This matrix includes all first partial derivatives of the polynomials in our homotopy $H(\mathbf{z}; t)$. At a branch point, the Jacobian matrix becomes singular. This is a condition that does not allow the Jacobian matrix to be inverted. For Newton's method, the Jacobian matrix must be invertible so if we are at a branch point, Newton's method is not possible.

If we are very near a branch point, the Jacobian matrix will be ill-conditioned, sometimes called nearly singular. Being ill-conditioned is defined by the condition number, $\kappa(A)$, becoming increasingly large in magnitude. For instance, at a branch point the condition number is infinite. Ill-conditioning can lead to undesirable behavior and possibly divergence of Newton's method. This is caused by the instability when inverting a nearly singular matrix.

Second, the use of prediction methods using the Jacobian may not be able to accurately predict the next step if they are near a branch point. This is caused by the geometry near a ramification point. When two distinct solutions are approaching each other, this occurs very rapidly. These quick changes in the geometry make predictions difficult.

Currently, the program *Bertini* will adaptively change step size and increase precision of approximations to combat the ill-conditioned zones near branch points. This is a good solution given the work done by Wilkinson [35]. Shortening the step size allows for better predictions to occur, and increasing precision of approximations decreases the size of ill-conditioned zones. However, using adaptive multiple precision requires a significant increase in run time. For example, an example with 8 equations and 8 variables causes a 13.3-fold increase in run time when we move from fixed precision to adaptive multiple precision in 64 digits [7]. We wish to design a process that allows us to avoid singularities, thus decreasing the need to adaptively change precision and perhaps step size during a run.

Chapter 2

AVOIDING SINGULARITIES DURING HOMOTOPY CONTINUATION

Given a set of polynomials $\mathbf{f} = \{f_1, f_2, ..., f_k\} \in \mathbb{C}[\mathbf{z}]$, we want to find the common solutions to the polynomials of $\mathbf{f}$. To do so, we construct a start system, $\mathbf{g}$, as in Section 1.2. We can then construct the homotopy, H, as in Section 1.3, i.e.,

$$\mathbf{H}(\mathbf{z}; t) = \mathbf{f}(\mathbf{z})(1-t) + t\mathbf{g}(\mathbf{z})$$

Consider the example,

$$\begin{aligned} h_1(x, y, t) &= (x^3 - 1) \cdot t + (x^3 + 2) \cdot (1 - t) \\ h_2(x, y, t) &= (y^2 - 1) \cdot t + (y^2 + 0.5) \cdot (1 - t) \end{aligned} \tag{2.0.1}$$

This example first appeared in [31]. For almost all values of $t \in \mathbb{C}$, there are six solutions. There are at least two branch points, though. In particular, there are two triple roots at $t = \frac{2}{3}$ and three double roots at $t = \frac{1}{3}$. We assume that we are unaware of these branch points *a priori*.

Our goal is to step through parameter space $\mathbb{C}$ from $t = 1$ to $t = 0.1$ along a path η that avoids as many branch points as possible.[1] There are several variants of this problem. For example, one could seek the shortest possible path that stays at least ϵ from each branch point for some small positive real number ϵ selected by the user or determined by the characteristics of the homotopy. Since the determination of such an ϵ is a very difficult

[1] We seek to reach $t = 0.1$, not $t = 0$, as endgames [4, 28, 29] begin operating at $t = 0.1$ or some other value of t away from 0, defined by the user.

problem (see Chapter 4), we instead seek to find as many branch points as we can in a reasonable amount of time and choose a piecewise linear path that avoids all discovered branch points by as far as possible.

2.1. LOCAL OPTIMIZATION

We employ the use of local optimization to find possible branch points. The particular method we use is called *gradient descent* [23, 24] though many alternatives would suffice. To use the method of gradient descent, we need to define an objective function. We introduce an objective function that has a minimum at branch points. We construct the objective function by taking all the polynomials in homotopy H and adding a few more homotopy polynomials that indicate when our Jacobian is singular. These additional polynomials come from the method in [3]. The matrix A in [3] will be our Jacobian matrix. In particular, we have the system

$$\widetilde{H} := \begin{bmatrix} H(t;\mathbf{z}) \\ \mathbf{J}(H(t;\mathbf{z})) \cdot B \cdot \begin{bmatrix} 1 \\ \xi_2 \\ \vdots \\ \xi_n \end{bmatrix} \end{bmatrix}. \tag{2.1.1}$$

This addition has added n equations since our Jacobian is an $n \times n$ matrix and we have also added $n-1$ new indeterminants, denoted ξ_i for $i = 2, ..., n$. These new indeterminants will not increase the number of paths to track, since each equation added is linear in the ξ_i. The matrix B is restricted to be a random unitary matrix so that B is numerically well conditioned.

If the determinant of $\mathbf{J}$ is zero, $\mathbf{J}$ is singular. Recall from Section 1.4 that when the Jacobian is near singular we may be near a branch point. Adding a single homotopy polynomial that is the determinant of the Jacobian may seem better than adding n homtopy polynomials in n new determinants. This is actually false as the determinant is usually a high degree polynomial that can be hard to compute, especially for large examples. In contrast, the method introduced in [3] adds n new equations, but they are all degree 1 in the new indeterminants and lower degree than the original equations in the original variables.

The entire set of polynomials is then

$$\{h_1(\mathbf{z};t), h_2(\mathbf{z};t), ...h_n(\mathbf{z};t), j_1(\mathbf{z},\xi;t), ..., j_n(\mathbf{z},\xi;t)\},$$

where h_i are the original homotopy polynomials and j_i are the additional homotopy polynomials in the system (2.1.1), ξ is a set of new indeterminants. We may then construct our objective function

$$OF(\mathbf{z};\xi;t) = \frac{1}{2}\left(h_1^2 + h_2^2 + \cdots h_n^2 + j_1^2 + j_2^2 + \cdots + j_n^2\right). \tag{2.1.2}$$

Near a branch point of $H(\mathbf{z};t)$, $OF(\mathbf{z};\xi;t)$ will have a minimum of 0. The converse is not true, we may find minima of $OF(\mathbf{z};\xi;t)$ that are not solutions to $H(\mathbf{z};t) = 0$.

2.2. CHOOSING A BOX

We first choose a rectangle R in the complex plane having $t = 1$ as the midpoint of the right side, height Δt_i, and width Δt_r, as depicted in Figure 2.1. The choice of rectangle dimensions is heuristic; we set $\Delta t_r = 0.1$, and $\Delta t_i = 0.2$ for now. Our goal is to rapidly find as many branch points as possible within this box. To do so, we shift the solutions at $t = 1$ by a user-defined factor $\mu\Delta t_r$ into the box, to a point we call t', and run local optimization methods from each of those solutions in an attempt to find nearby ramification points, which are easily projected to branch points in $\mathbb{C}$. The central idea is that for $t' \approx 1$ (i.e., Δt_r and γ not too large), the solutions of $\mathbf{H}(\mathbf{z}, t)$ at $t = 1$ should be near the solutions at $t = t'$.

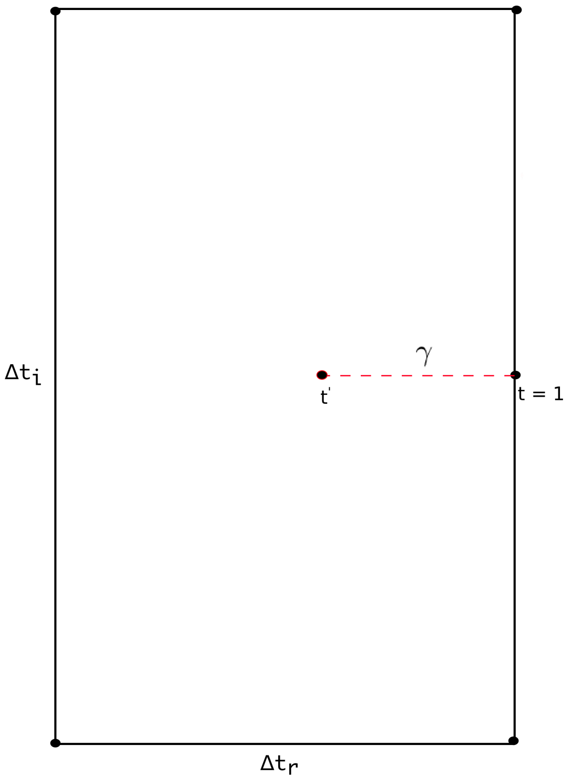

FIGURE 2.1. Rectangle in which branch points are sought.

2.3. FINDING BRANCH POINTS IN RECTANGLE R

We can use local optimization methods along with the objective function (2.1.2) created in Section 2.1 to search for branch points within box R. We constrain our optimization problem to the interior of R; minimizers of $OF(\mathbf{z}; \xi; t)$ outside R are irrelevant in this step.

Notice that this optimization step could produce false positives. Indeed, a minimum $(\mathbf{z}^*; \xi^*; t^*)$ of objective function $OF(\mathbf{z}; \xi; t)$ need not satisfy $\mathbf{H}(\mathbf{z}^*, t^*) \approx 0$. Any computed potential minima could be checked for this condition as well.

It is important to remember that this method is intended as a fast heuristic, not a complete algorithm. There is no guarantee that all branch points within R will be found. In fact, it is possible for there to be more branch points inside R than start points for optimization. However, if we can avoid some ill-conditioned zones, time may be saved during path tracking. Since not all branch points may be found, it is advisable that this method be paired with an adaptive precision method to work through any undetected ill-conditioned zones.

2.4. CREATING WAYPOINT W_2

Once local optimization has terminated, we have a (possibly empty) collection of approximations of branch points within the rectangle R. The task now is to choose a new waypoint w_2 to be connected to the first waypoint $w_1 = 1$ while avoiding as many of these potential branch points as possible. It is obvious that the length of the path for t contributes to longer runtime, so it is best not to stray farther than necessary from the real line. To partially mitigate this concern, we set up two guards that ensure that we only move to the left, in the general direction of $t = 0.1$. Figure 2.2 shows two such guards. In the case that the optimization phase produces no approximate branch points, w_2 should be chosen in the direction of $t = 0.1$, the ultimate target of path η.

Figure 2.3 illustrates how we choose to navigate through the box from waypoint w_i to waypoint w_{i+1}. If $\Delta t_i = .2, \Delta t_r = .1$, then our guards are $\pm\frac{\pi}{4}$ radians from the horizontal line extending to the left from the waypoint. Once the guards are in place, we can compute angles between the guards and the nearest branch point. We continue this process for nearest neighbors of possible branch points, decided by the line segment from the current waypoint to the possible branch point. Once this is done, we can compute the largest angle between

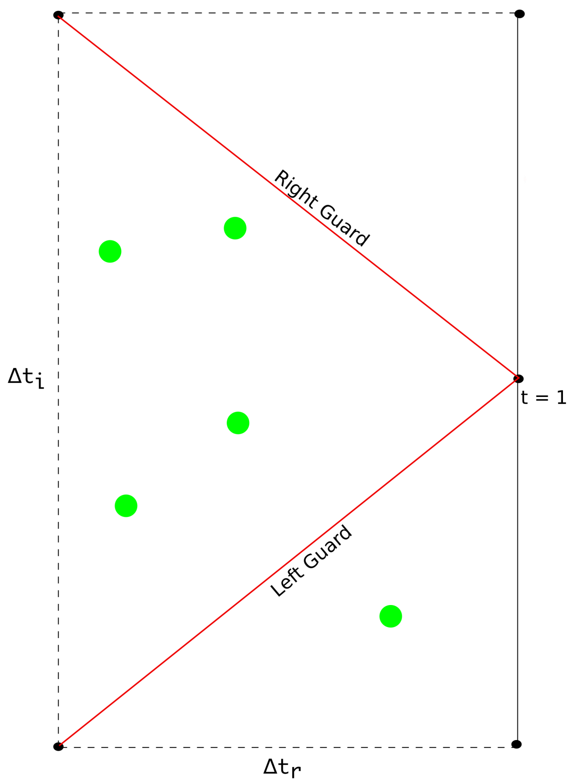

FIGURE 2.2. Rectangle R after optimization with minima (dots) and guards (labeled line segments).

nearest possible branch points. In Figure 2.3 we see that θ_3 is the largest angle. We take this largest angle and bisect it. We move to the next waypoint by following all solutions from the previous waypoint, using standard continuation.

How far do we wish to move in this direction? The width of the box, Δt_r, is not optimal because we do not know if there are branch points right outside of the box. For this reason, we move a fraction of the width, $\mu \Delta t_r$ where $\mu < 1$. This is a user defined tolerance, with default set to $\mu = \frac{3}{4}$. We call this new point, waypoint w_2. We can now repeat the process going from waypoint w_i to waypoint w_{i+1} until we get to $t = 0.1$.

What if our path does not lead to $t = 0.1$? This is a valid concern. If we find no branch points in a box, we move in the direction of $t = 0.1$. At the end, when the real part of t is 0.1, we can always take one more piecewise linear step from $t = 0.1 + ai$, where $a \in \mathbb{R}$, to

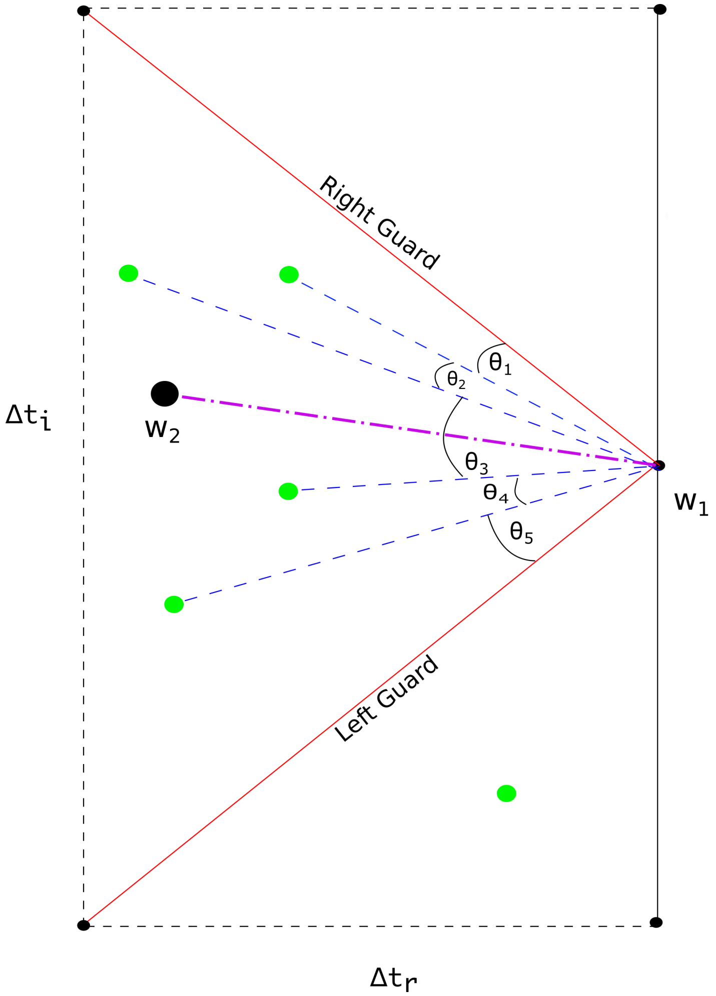

FIGURE 2.3. We start with solutions at the waypoint w_1. We shift them and use local optimization to find the green points that are possible branch points. We next want to find a way to avoid the branch points. Starting with making the left and right guards to make sure we move toward $t = 0.1$, we compute the angles between nearest neighbors and the guards. We take the maximal angle θ_3 and bisect that angle. This angle will stay as far as possible from detected possible branch points.

$t = 0.1$. An alternative method would be to weight the decision to move towards $t = 0.1$ as more desirable as we progress through the procedure. At the current time are details that are left to future work.

2.5. PSEUDOCODE AND DETAILS

As described above, the point of avoiding ill-conditioned zones around branch points is to save computational resources, with the aim of making the whole run more efficient. Of course, it would be counterproductive to use massive computational resources to construct path η. A guiding principle is thus to choose η on the fly as rapidly as possible. Naturally, one must weigh efficiency against certainty. A rapid, careless search saves η construction time but may miss branch points; a slow, careful search takes more η construction time but will also catch more branch points. Finding the optimal balance remains an open problem, and is surely problem and user dependent.

This heuristic contains various choices that can easily be replaced. For example, it might be beneficial to choose segments of η differently, to use a different objective function, to use different optimization methods, etc. Also, the various tolerances of the heuristic are all user-defined and could possibly be made adaptive. The choices in the pseudocode seemed reasonable, if not optimal, as we investigated this technique.

On a similar note, our method synchronizes the tracking of all paths. It might be better to choose η for one path or a small batch of paths at a time. In that case, one must be careful of handling monodromy appropriately.

In this section, we provide a formal statement of our method in pseudocode, followed by remarks about various details.

`Heuristic to determine path` η

Input: Homotopy $H(\mathbf{z};t)$; m solutions S of $h(\mathbf{z};1) = 0$; (optionally) parameters $\Delta t_r, \Delta t_i$ defining search boxes.

Output: Piecewise linear path η from $t = 1$ to $t = 0.1$, given as a sequence of waypoints w_i, $i = 1, \ldots, k$.

```
Set $w_1 = 1$.
Set $k = 1$.
Set $\Delta t_r = 0.1, \Delta t_i = 0.2$ if not provided.        ▷ Fixed here; could be made adaptive.
while $t \neq 0.1$ do
    Set $t = w_k - \frac{1}{2}\Delta t_r$.                      ▷ $t$ is the midpoint of the box.
    Set $b = 0$.                                                ▷ Counts branch points discovered.
    for $i = 1, \ldots, m$ do
        Run optimization problem,                               ▷ $OF(\mathbf{z};\xi;t)$ defined in equation 2.1.2
                $\underset{\mathbf{z},\xi,t}{\text{minimize}} \quad OF(\mathbf{z};\xi;t)$
                $\text{subject to} \quad t \in X$
        starting from $i^{th}$ point of $S$.                    ▷ Box $X$ constructed from $w_k, \Delta t_r, \Delta t_i$
        if Convergence to a branch point within the box determined by $\Delta t_r, \Delta t_i$ then
            Store minimum in $B$, increment $b$.
        end if                                                  ▷ Ignore runs that leave the box.
    end for                                                     ▷ Now have some branch points within the box.
    Order B based on imaginary part.
    for $j = 1, \ldots, b - 1$ do
        Find angle $\alpha_j$ between $B[j]$ and $B[j+1]$, measure from $w_k$.
    end for
    Set $J = j$ such that $\alpha_j > \alpha_i \forall i \neq j$.     ▷ Max angle between branch points.
    Let $\mathbf{v}_J$ be the vector that makes the angle $\alpha_J$ with the horizontal.
    Increment $k$.
    Set $w_k = w_{k-1} + \frac{\mu \Delta t_r}{cos(\alpha_J)}\mathbf{v}_J$
                                                                ▷ If $B = \phi$ then $\mathbf{v}_J$ is the vector from $w_{k-1}$ to 0.1
    Reset $B = \phi$.
end while
```

2.6. OUT PROOF-OF-CONCEPT IMPLEMENTATION

We implemented this heuristic using a combination of MATLAB [25] and Bertini [5]. To interface between these, we used the MATLAB package BertiniLab [9]. In addition, all settings, e.g., rectangle dimensions, shifting factor, and start systems, are controllable by the user. The use of Matlab and BertiniLab significantly adds to the computation time, compared to a Bertini-only run. In particular, each time a set of paths is tracked, there is mathematically unnecessary significant overhead with the reading and writing of files. With the ongoing redevelopment of Bertini as Bertini 2.0, we intend to implement this technique natively, surely giving much better run times. The implementation of this heuristic is part of a bigger project that will be discussed in Section 3.1.

2.7. EXAMPLES

Recall Example 2.0.1 from Chapter 2.

$$\begin{cases} h_1(x,y,t) = & (x^3-1)\cdot t + (x^3+2)\cdot(1-t) \\ h_2(x,y,t) = & (y^2-1)\cdot t + (y^2+0.5)\cdot(1-t) \end{cases} \tag{2.7.1}$$

There are at least two branch points, in particular, there are two triple roots at $t = \frac{2}{3}$ and three double roots at $t = \frac{1}{3}$.

One run of an implementation of our heuristic yielded Figure 2.4. Notice that we pick up two minima around $t = \frac{2}{3}$. This pushes our piecewise linear path below the real axis. Once we are beyond the minima, we start moving toward $t = 0.1$. Around $t = \frac{1}{3}$ we do not encounter any minima. This may be a sign that we are too far from the minimum to notice the ill-conditioned zone around $t = \frac{1}{3}$. One must keep in mind that this heuristic has a random unitary matrix in the construction of the objective function 2.1.2. For this reason, another run of our heuristic would give us a different path.

2.7.1. NEURAL NETWORK EXAMPLE. Another example comes from [32]. Explicitly, the homotopy is,

$$
\begin{cases}
h_1(x,y,z,t) = (xy^2 + xz^2 - \frac{11}{10}x + 1)\cdot t + (x^3 - 1)\cdot(1-t) \\
h_2(x,y,z,t) = (yx^2 + yz^2 - \frac{11}{10}y + 1)\cdot t + (y^3 - 1)\cdot(1-t) \\
h_3(x,y,z,t) = (zx^2 + zy^2 - \frac{11}{10}z + 1)\cdot t + (z^3 - 1)\cdot(1-t).
\end{cases}
\tag{2.7.2}
$$

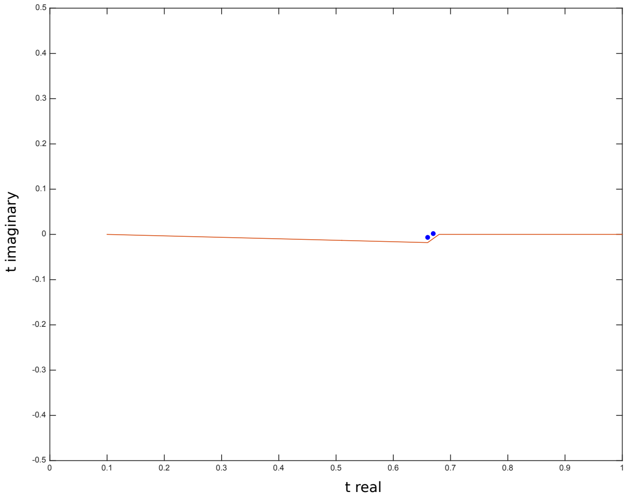

Figure 2.4. A run using Example 2.0.1.

This system has at most 42 points according to a run using *Bertini*. This system should cause a more turbulent path to be constructed, and indeed it does. In fact, the solution set at $t = \frac{1}{2}$ has a multiplicity 84 solution. If we look at Figure 2.5 we see that there is definitely a detection of minima around $t = \frac{1}{2}$. Once beyond $t = \frac{1}{2}$, we attempt to move back towards $t = 0.1$, and encounter one more minimum before being able to construct a path straight to $t = 0.1$.

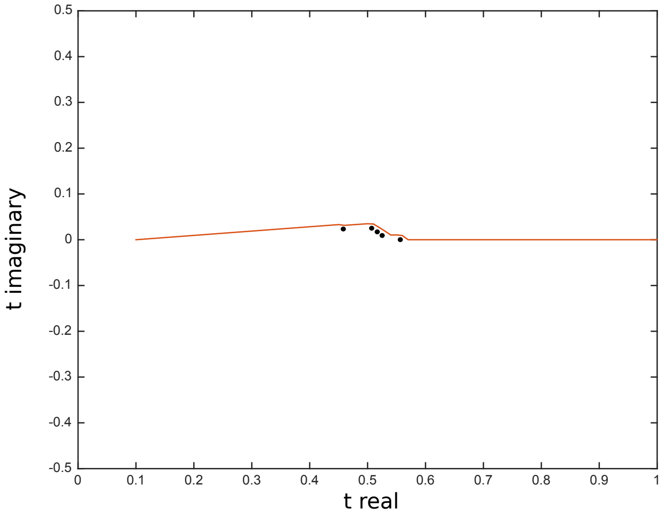

FIGURE 2.5. Illustration of path formation for Example 2.7.2 using our heursitic.

2.7.2. ADJACENT MINOR SYSTEMS. A final example to illustrate the procedure outlined in Chapter 2 is the set of adjacent minor systems. To construct an adjacent minor system, we first consider a matrix of size $2 \times n$. Let this matrix be filled with indeterminants, as shown below.

$$\begin{bmatrix} x_1 & x_2 & x_3 & \dots & x_n \\ x_{n+1} & x_{n+2} & x_{n+3} & \dots & x_{2n} \end{bmatrix}$$

We take as our polynomials the 2×2 adjacent minors of this matrix, using only columns adjacent to each other. This yields $n-1$ equations in the $2n$ variables explicitly stated in equation 2.7.3. Notice that this construction has given us a non-square system, particularly an underdetermined system with more variables than constraints. This therefore has a positive dimensional solution set. Adding random linear equations, we can make this system square. This reduces our solution set to isolated solutions.

$$(2.7.3)\qquad \begin{cases} f_1 = x_1x_{n+2} - x_2x_{n+1} \\ f_2 = x_2x_{n+3} - x_3x_{n+2} \\ f_3 = x_3x_{n+4} - x_4x_{n+3} \\ \vdots \\ f_{n-1} = x_{n-1}x_{2n} - x_nx_{2n-1} \end{cases}$$

This system has been studied and has a complete intersection with codimension is $n-1$ and degree is 2^{n-1} [22]. The linear equations we add are in the same variables and are denoted $L_1, L_2, ..., L_{n+1}$. For ease, we will consider $\mathbf{x} =< x_1, x_2, ..., x_{2n} >$.

We can now construct a homotopy system,

$$(2.7.4)\qquad \begin{cases} h_1(\mathbf{x}) = f_1(\mathbf{x}) \cdot t + (x_1^2 - 1) \cdot (1-t) \\ h_2(\mathbf{x}) = f_2(\mathbf{x}) \cdot t + (x_2^2 - 1) \cdot (1-t) \\ h_3(\mathbf{x}) = f_3(\mathbf{x}) \cdot t + (x_3^2 - 1) \cdot (1-t) \\ \vdots \\ h_{n-1}(\mathbf{x}) = f_{n-1}(\mathbf{x}) \cdot t + (x_{n-1}^2 - 1) \cdot (1-t) \\ h_n(\mathbf{x}) = L_1(\mathbf{x}) \cdot t + (x_n - 1) \cdot (1-t) \\ h_{n+1}(\mathbf{x}) = L_2(\mathbf{x}) \cdot t + (x_{n+1} - 1) \cdot (1-t) \\ \vdots \\ h_{2n}(\mathbf{x}) = L_{n+1}(\mathbf{x}) \cdot t + (x_{2n} - 1) \cdot (1-t) \end{cases}$$

Figures 2.6 and 2.7 are runs where the matrix we consider is 2×3. The output in Figure 2.7 shows that there is a consistent area near $t = 0.8$ that is causing us to change our path. We also encounter a possible singularity that was not detected during our first run. Notice that we do not avoid this one as well. This could be because it was near a waypoint that we had moved to in a previous step.

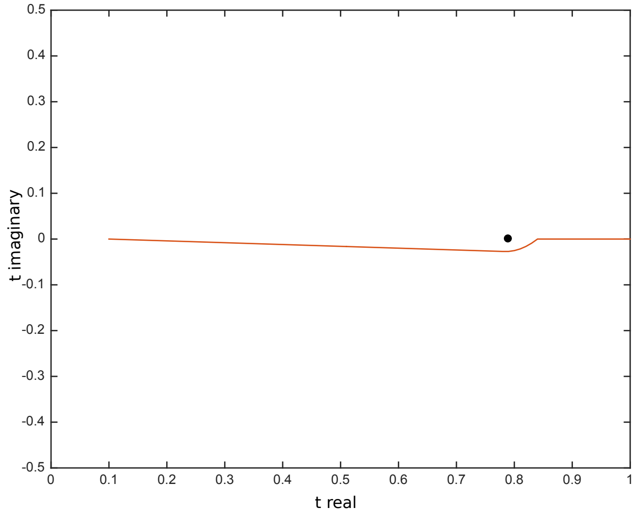

FIGURE 2.6. Adjacent minor run using a 2×3 matrix of indeterminates.

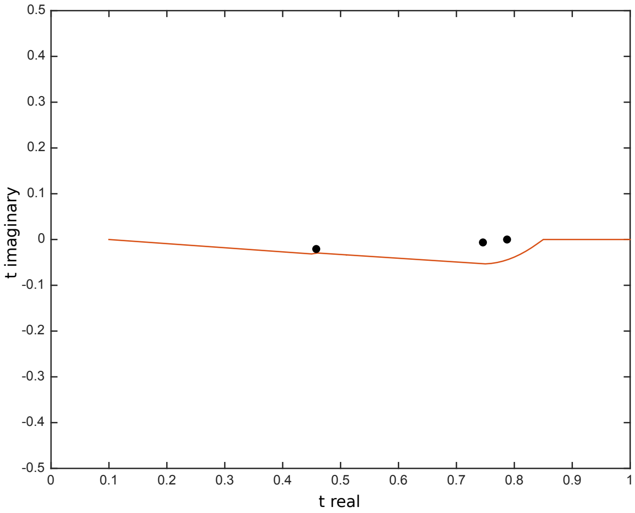

FIGURE 2.7. A second run of the adjacent minor system using a 2×3 matrix of indeterminates.

2.8. DISCUSSION OF METHOD OF AVOIDING BRANCH POINTS

The primary benefit of this new method is the ability to avoid some ill-conditioned zones during homotopy continuation. Ill-conditioning leads to higher precision or, alternatively, inaccuracy. The former can be expensive; the latter is dangerous.

As the number of variables increases, it may become harder to find minima that may be branch points. This is caused by the fact that even if all but one of the coordinates match a branch point exactly, we still will have issues converging to that point. This can be seen looking at Example 2.6 compared to Example 2.7. The method finds points to avoid, but seems to find fewer in Example 2.6. As the number of variables increases, we are then trying to converge in a larger space. Also, as the number of variables and the number of equations increase we will see a slow down in run time, as is the case with any method. Much of this has to do with the synchronization of all the paths to a specific t value. The more paths there are to move, the longer it will take to move all of the paths.

Another slow down is the interfacing between Bertini [5] and MATLAB [25]. Much of the interfacing currently requires the use of reading and writing to files. This is unnecessarily slow. In Section 3.1 we discuss a plan to allow all interfacing to be done in Bertini. This would eliminate the use of MATLAB and will allow us to streamline all the heuristics into a seamless package. With the work described in Section 3.1, it may also be feasible to avoid synchronizing paths.

Problem	Tot. Time	Optim. Time	Bertini/Communication Time
Noon3	428	270	154
Noon4	637	114	517
Adjacent Minor (2x3)	136	40	92
Adjacent Minor (2x6)	1098	45	1026
Adjacent Minor (2x9)	16985	2253	14658

FIGURE 2.8. Average run times for running our proof-of-concept implementation on five examples several times each, in seconds.

2.8.1. TIMING AND SCALING. Figure 2.8 shows us that the overall time is dominated by two operations: optimization and communication time between Bertini and MATLAB. Optimization time is the time added by the new method of this paper. Notice that it seems to scale relatively well, especially in comparison to the inescapable Bertini run times. Bertini run times are also expected to decrease significantly with the development of Bertini 2.0. Communication time will be cut to 0 when this method is implemented natively within Bertini 2.0. Thus, comparison of these run times to existing packages is misleading; this new method, as currently implemented, cannot compete.

The question of how useful this method will be once it is fully operational cannot be decided until Bertini 2.0 is completed in the coming years. For now, this method is intentionally left at the proof-of-concept stage, to be more carefully studied, modified, and optimized once a thorough implementation is feasible.

2.9. OTHER METHODS

There are currently four methods for handling branch points during homotopy continuation related to this method: Using adaptive precision (the current default in Bertini), not using adaptive precision (the default in all other numerical algebraic geometry software packages), finding *all* branch points, and the monodromy technique of [31]. This thesis proposes a fifth.

Finding all branch points for a homotopy $\mathbf{H}(\mathbf{z}, t)$ is a harder problem than finding the solutions of $\mathbf{H}(\mathbf{z}, t)$ itself. Indeed, one can write down a polynomial system that has among its solutions all branch points of $\mathbf{H}(\mathbf{z}, t)$. In Section 2.1 we discuss one way to create such a polynomial system. However, this system has more polynomials and variables than $\mathbf{H}(\mathbf{z}, t)$ and comes with the intrinsic problem that it, too, requires the handling of branch points in some way. Furthermore, not all branch points are needed! We care only about those near the real line, between $t = 0$ and $t = 1$, at least for a standard, straight-line homotopy.

Instead of relying on homotopy continuation to solve a problem related to homotopy continuation, one could employ exact methods, e.g., those based on resultants, to find all branch points. In fact, we first experimented with this approach but found it to be prohibitively inefficient, though we admittedly used black box methods in a relatively naïve way. Once we realized that we don't even care to know about branch points in the vast majority of $\mathbb{C}$, we abandoned this method in favor of local methods.

Finally, as in [31], one could use the condition number as a signal for the presence of an ill-conditioned zone. The method of this paper and the method of [31] are both heuristics, with various benefits and limitations, and the optimal method might be some combination of the two. A decision on this can be made only after a careful implementation of all options has been completed.

Chapter 3

OTHER PROJECTS

All software reaches a point where redevelopment becomes desirable. Reasons can include limitations resulting from language choice, inflexibility of the original design, or external forces such as demands beyond the scope of the original design. Bertini is currently under redevelopment for many reasons, some of which are mentioned above.

3.1. BERTINI 2.0 DEVELOPMENT

For my professional development, I decided to become a developer for the software Bertini 2.0. The skill set gathered by becoming a developer is diverse and has helped me start a career in software development after my degree is completed. Also, by directly implementing algorithms that I use on a daily basis, I am allowing myself to see possible optimizations. These algorithm and software optimizations are aimed at making homotopy continuation as computationally efficient and robust as possible.

Before redeveloping or creating any software package there should be clear preparations to reduce the risk of wasting time and resources [27]. Clearly defining the requirements of the software will make sure you are solving the correct problem when constructing the software. The Bertini 2.0 team has clearly defined requirements for redevelopment. The main goal is to allow Bertini 2.0 to be flexible, modular, and extendable for the foreseeable future. We must allow backwards compatibility and ensure that Bertini 2.0 is user friendly. Also, a developer goal is to reduce the duplicate code that exists inside of the original implementation. I redeveloped the endgame algorithms inside of Bertini with these goals in mind.

The two endgames in particular are the Cauchy endgame and the Power Series endgame. There are many references that explain these two algorithms; see [8, 28, 29]. To achieve flexible, modular code, we use object-oriented programming and make the Cauchy Endgame and Power Series Endgame into derived classes of a general endgame class. For more on object oriented programming, see [33].

In the original implementation of Bertini the endgames are responsible for tracking a path from $t = 1$ to the endgame boundary, before actually executing the endgame. To be modular and flexible, we remove the tracking to the endgame boundary from the endgames. This makes the endgames a modular unit that is specifically used for the end of homotopy continuation. In addition, the original implementation had duplicate code to allow for double precision arithmetic or multiple precision arithmetic. In the redevelopment, the different endgame classes are templated based on the type of numbers that have been used to track to the endgame boundary. This reduces the duplicate code and also increase readability for the users and developers. It should be noted that templating increases compile time but is a one time compilation step. The Bertini 2.0 team has agreed that this is acceptable.

The use of classes and object-oriented programming allows there to be specific objects that hold all the algorithms needed for computation. Then, using the idea of *inheritance*, there are derived classes that are specific for certain types of endgames. Specifically, the fixed precision endgames and the adaptive precision endgames. These changes in structure allow the endgames to hold their “state” in member variables, and can allow for outside functions to observe and trigger events outside of the endgames. The use of observers and events is crucial for the problem stated in Section 3.2.

Up to this point, we have discussed how to reduce duplicate code to make Bertini 2.0 flexible. To allow user friendliness and backwards compatibility, the Bertini 2.0 team has adopted the paradigm of *test driven development* (TDD). This paradigm requires a developer to design test cases for every function that they create. This allows for stable development. If a change to the code causes a set of cases to fail, either the tests are no longer correct or the change has altered existing functionality. In addition, a new user can look at the test cases to find out how to use the functions. The use of test cases also allow Bertini 2.0 to be checked against Bertini 1.0 and make sure we have backward compatibility.

The implementation of the endgames in Bertini 2.0 has been an invaluable experience to me. I have realized I enjoy computer programming and would like to have a career where my mathematics background and joy of programming allow me to help create unique algorithms and implementations to solve challenging problems. In particular, the application of local optimization to avoid branch points during homotopy continuation can be directly implemented into Bertini 2.0. Many of the issues concerning this method of avoiding branch points comes from the use of files and that the handling of data is clumsy due to implementation diffiiculties using MATLAB [25, 9]. Both of these issues will be overcome by implementing my heuristic in Bertini 2.0. In Bertini 2.0, all data can be held in memory and can be handled appropriately to streamline the use of homotopy continuation. I look forward to implementing the heuristic in this manner and comparing the resulting capability of my heuristic.

3.2. AUTO TUNING

When running Bertini, there are many warnings and messages that one can encounter. Bertini will warn about potential path crossings and will remark on path failures. These path failures can happen by reaching maximum precision while tracking a path or by tracking to a minimum time value. As a user, these warnings and messages can be frustrating and difficult to remedy. Along with warnings, users can encounter slow performance and wish to increase the speed at which Bertini solves their particular problem.

Fortunately for the Bertini user, there are user-defined settings to mitigate any issues that arise. The unfortunate aspect is a user may not know which settings to alter to alleviate a specific warning/problem they are encountering. In Appendix E of [8] there is a list of 69 user-defined settings. Nineteen of these settings come with a finite number of choices, for example predictor choice, leaving the user with around 1.2 billion combinations to choose from. The other 50 settings are floating point tolerances or other discrete settings that are theoretically infinite in their number of combinations.

The problem of picking user defined settings can be daunting one the inexperienced user. A user's best solution is to consult with an expert, but there are very few experts and their time is limited. This causes a major bottleneck and the user is left waiting to hear back from an expert. It would be extremely useful if Bertini could decide some settings to be more optimal for the given problem a user wishes to solve. There is, however, no analysis on the possible settings or heuristics that would best suit a certain problem. Some users try their best to choose settings, but whether or not these choices are optimal (or even reasonable) is impossible to know.

The problem is more complex than just finding the optimal settings for a polynomial system. First, not all polynomial systems are created equally. Depending on the structure of the system, and how the user wants to solve the system, the settings that are optimal for one problem may be completely different than those from another problem. Different users may prefer speed over security. The sheer amount of combinations of settings makes it not entirely clear which settings to choose. Also, there are many other aspects of solving polynomial systems that need to be considered. How does one certify an answer or validate/verify the solving of a polynomial system?

We have started to build an auto-tuning tracker to solve this complex problem. It is clear that many decisions must be made. What data should be used in decision making for an auto-tuning tracker? What problem sets would this be best used on? Are there user defined settings that are negligible? We are starting by collecting data and analyzing condition numbers during tracks to best sort singularity issues. An auto-tuning tracker could use the method from Chapter 2 to decide on what direction to move during homotopy continuation.

The implementation of an auto-tuning tracker also ties back to the project in Section 3.1. During redevelopment, we can create an infrastructure of observers to watch for certain events during homotopy continuation. Observers are objects that can watch and react to events.This will allow for seamless interfacing during homotopy continuation to determine when settings should be changed.

3.2.1. AUTO-TUNING AWAY FROM $T = 0$. This project is split into two parts, the study of homotopy continuation away from $t = 0$ and that around $t = 0$. These two areas are studied independently because each of these areas use different algorithms. For instance, homotopy continuation around $t = 0$ uses the endgames [8]. Most data collection and work has been done on homotopy continuation away from $t = 0$.

My collaborators and I decided to look at a subset of settings that a user can set during a *Bertini* run, namely: prediction method, tracking tolerance, and number of Newton iterations. These settings are commonly-adjusted and serve as a basis to start our analysis.

In Section 1.3 we discussed basic homotopy continuation and how you need to choose a prediction method. By using prediction method as one of our settings, we can see how lower and higher order predictors perform. Tracking tolerance is a setting that allows us to choose how small our Newton residual must be to say that a predictor-corrector step is successful. Number of newton iterations is how many times we are allowed to run newton iterations to try to correct back to the path. The dependence of run time on the choice of random numbers prompts us to run each set of settings five times with different random seeds inside of *Bertini*. This allows us to look past minor inconsistencies that may be due to picking a bad random seed.

We have so far used two polynomial systems in particular. These systems are smaller and have been used to calibrate the settings we will use in the future to study larger systems. The first system has been coined the "illustrative example." It illustrates the different types of solution sets one can find in numerical algebraic geometry as its solution set consists of an isolated point, a curve, and a surface. This system, in input file format, can be seen in the Appendix A.

The second system is discussed in detail in [8]. This system is known as the "Lagrange points problem." This is because it originates from Lagrange's paper that gives the exact solution to the three-body problem of gravitation. The input file format of this problem can be seen in Appendix A. The Lagrange points problem has been observed to be more sensitive to settings in *Bertini* than the illustrative example.

These two systems are the starting point for our data collection in the Auto-Tuning project for Bertini 2.0. The plots in Figure 3.1 and Figure 3.2 display the data collection on both systems described above.

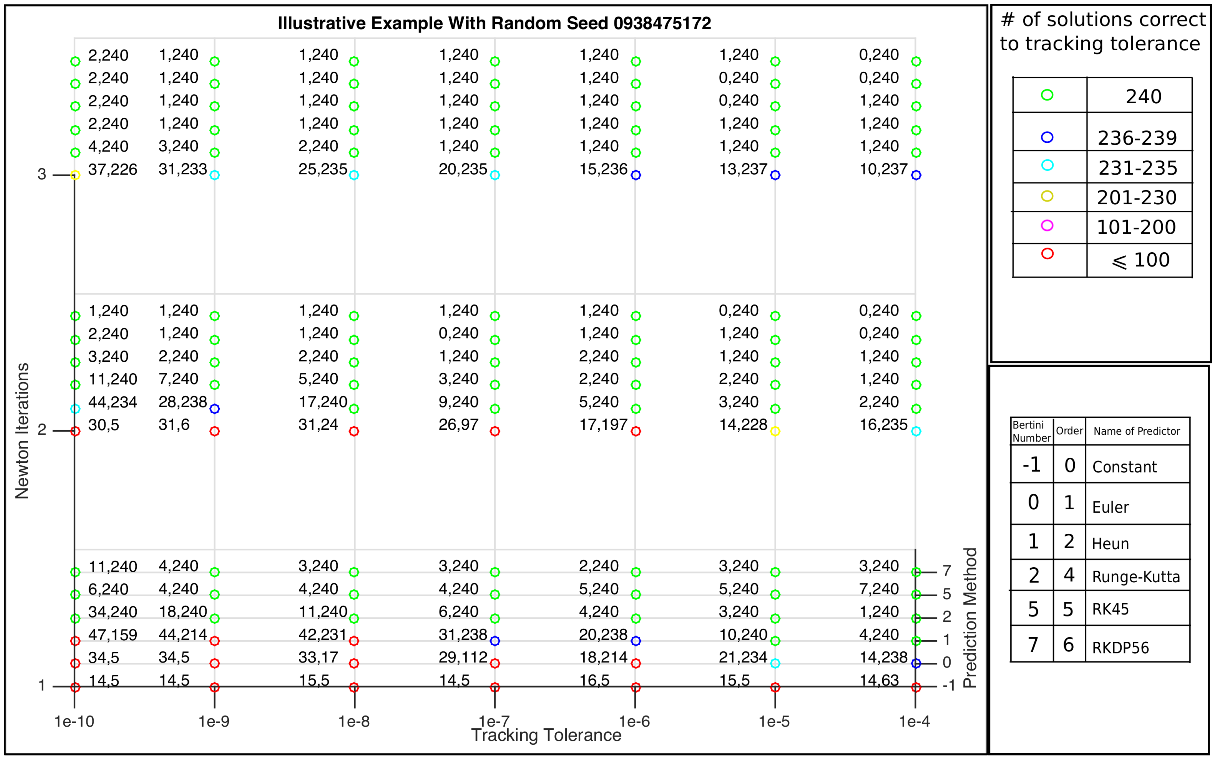

Figure 3.1. Lattice where each point in the lattice corresponds to a run with a different group of settings. The color depicts how many solutions we have correct up to the tracking tolerance we picked. The two numbers by each lattice point depict run time and number of solutions that were not affected by path crossings respectively.

3.2.2. CONCLUSIONS AWAY FROM $T = 0$. There are some conclusions drawn from the data collected from the illustrative example and Lagrange points problem. First, we should keep the number of newton iterations below 3. From our discussion in Section 1.3, Newton iterations are used to correct back to the path in homotopy continuation. It may seem beneficial to let Newton's method run until it converges or diverges, but this can be risky. If we happen to take a prediction step into a different Newton convergence basin and we take too many Newton iterations, we can get a path crossing. For this reason it is better to allow failure of our predictor-corrector scheme. Allowing failure will reduce the possibility to path crossings to the case when paths are very close to each other. In Section 1.4 the idea of adaptive stepsize is mentioned, and is another feature to reduce the likelihood of

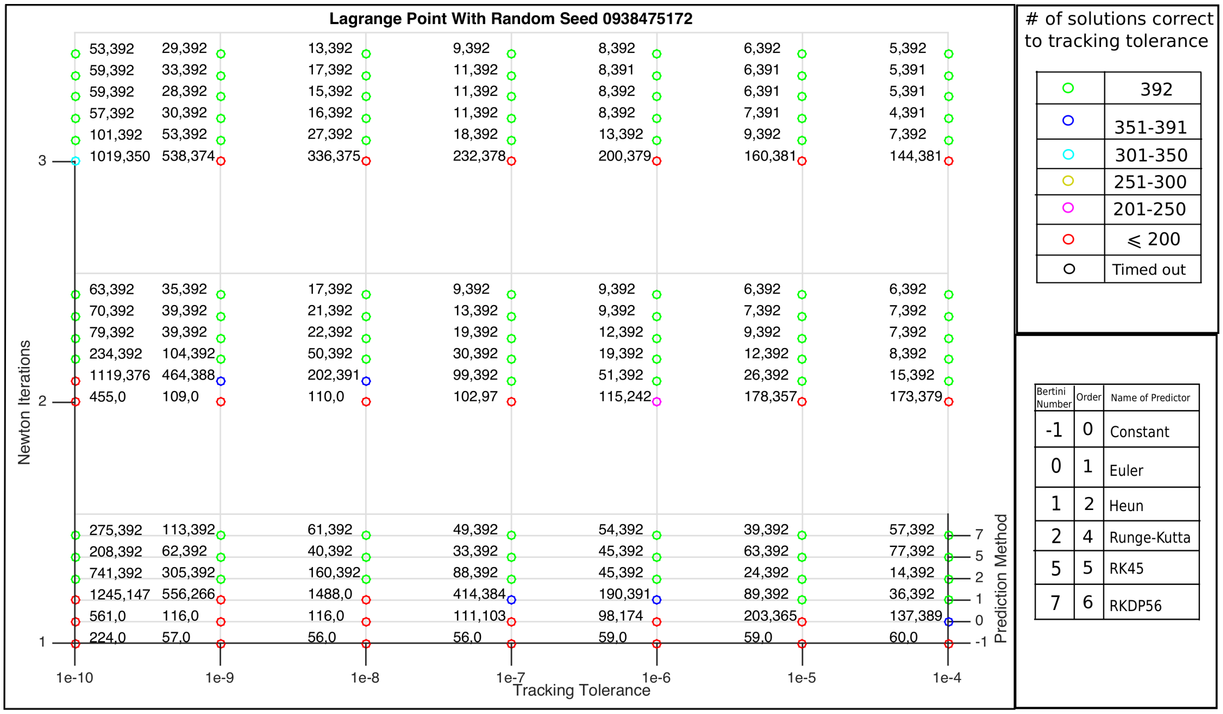

FIGURE 3.2. Each point in this lattice corresponds to a run with a different group of settings. The color depicts how many solutions we have correct up to the tracking tolerance we picked. The two numbers by each lattice point depict run time and number of solutions that were not affected by path crossings, respectively.

path crossings. However, when we allow only one Newton step we have many issues. It can be very slow, because with only one Newton iteration we will quite often fail to converge. Failing to converge means we take shorter steps with adaptive stepsize, and this will cause many paths to fail from reaching maximum number of steps allowed.

Also, from Figures 3.1 and 3.2, we see that the constant predictor is inefficient. Most predictors like Runge-Kutta and its variants use information from the Jacobian matrix at a certain point. For more on the specifics of these prediction methods, please consult [23]. The constant predictor does not use this information and instead attempts to move only the t value and leave all other values constant. One caveat is that the constant predictor is still a very useful option if a user wishes to use certification methods [19, 20]. To certify, the algorithm must be able to assert validity at each step of the homotopy continuation algorithm. This is only achieved with the constant predictor at the current time.

If we tighten our tracking tolerance and use lower order predictors, we see that we fail many paths. This has to do with the maximum number of steps we are allowed to use on any given path. A quick conclusion is that if you wish to use lower order predictors but strict tracking tolerance, you should increase the number of steps allowed for any given path. This will increase the success rate but will also increase run time.

The current analysis shows that using a loose tolerance with a higher order predictor with limited number of Newton iterations appears to be optimal, at least for these examples. This can be seen from Figures 3.1 and 3.2. We see a high rate of success with limited path crossings, and our run time is small. We must consider more examples with various structures and contexts to see if this is more generally the case.

3.2.3. OPEN QUESTIONS IN AUTO-TUNING. The analysis done so far is a good start but leaves many open questions.

(1) Are there other settings that we should look at with or without those that we have already explored?
(2) Are there other metrics of success that we should monitor?
(3) What problems should be considered for the next set of analysis?

Chapter 4

FUTURE WORK

This section details two problems to consider in the future. The author would encourage any reader to suggest ways of progressing on either project.

4.1. THE NEAREST SINGULARITY

The heuristic discussed in this paper has given a way to navigate during homotopy continuation in order to avoid branch points. After working on this heuristic, a decision that has to be made is how far to travel in a particular direction before doing another round of searching for singularities. The inverse of the condition number gives a relative distance to the nearest singular matrix [21]. However, it is not known how to find the direction to this nearest branch point. If this direction could be found, it would allow path determination from Section 2.4 to be more informed as it would provide nearest singularity to avoid.

4.2. ESTIMATING THE ENDGAME BOUNDARY

In *Bertini*, the default value for the endgame boundary is $t = 0.1$. This is the radius of the disk centered at the origin inside which we assume there are few or no branch points. Heuristically, this value is fine, but in practice, it turns out that there are usually branch points inside this disk. Convergence of the endgames requires that we be inside the endgame boundary [8]. For this reason, it would be desirable to know a better value, computed or heuristically found, for the endgame boundary.

The method for avoiding singularities may be of use for this application. Searching for singularities inside the disk could reduce the amount of computation needed to reach the actual endgame boundary for a specific run. In fact, if run on an application with a large number of paths, we may be able to discern a “best” endgame boundary for most paths. Not much is known about how large or small the endgame boundary can be. This project could produce a new way of looking at how the endgame boundary is chosen and may significantly decrease run time by avoiding useless endgame iterations.

BIBLIOGRAPHY

[1] E. L. Allgower and K. Georg. *Introduction to Numerical Continuation Methods.* SIAM, Philadelphia, 2003.

[2] M.A. Armstrong. *Basic Topology.* Springer-Verlag, third edition, 1983.

[3] D.J. Bates, J.D. Hauenstein, C. Peterson, and A.J. Sommese. Numerical decomposition of the rank-deficiency set of a polynomial matrix. *Approximate Commutative Algebra*, 1:1–22, 2010.

[4] D.J. Bates, J.D. Hauenstein, and A.J. Sommese. A parallel endgame. *Contemp. Math.*, 556:25–35, 2011.

[5] D.J. Bates, J.D. Hauenstein, A.J. Sommese, and C.W. Wampler. *Bertini*: Software for numerical algebraic geometry. Available at betini.nd.edu.

[6] D.J. Bates, J.D. Hauenstein, A.J. Sommese, and C.W. Wampler. Adaptive multiprecision path tracking. *SIAM Journal on Numerical Analysis*, 46:722–746, 2008.

[7] D.J. Bates, J.D. Hauenstein, A.J. Sommese, and C.W. Wampler. Stepsize control for adaptive precision path tracking. In *Interactions of Classical and Numerical Algebraic Geometry*, volume 496 of *Contemporary Mathematics*, pages 21–31. American Mathematical Society, 2009.

[8] D.J. Bates, J.D. Hauenstein, A.J. Sommese, and C.W. Wampler. *Numerically Solving Polynomial Systems with Bertini.* SIAM, 2013.

[9] D.J. Bates, A. Newell, and M. Niemerg. Bertinilab: A matlab interface for solving systems of polynomial equations. *Numerical Algorithms*, 71:229–244, 2016.

[10] Tianran Chen, Tsung-Lin Lee, and Tien-Yien Li. Hom4PS-3: A Parallel Numerical Solver for Systems of Polynomial Equations Based on Polyhedral Homotopy Continuation Methods. In Hoon Hong and Chee Yap, editors, *Mathematical Software – ICMS 2014*, number 8592 in Lecture Notes in Computer Science, pages 183–190. Springer Berlin Heidelberg, January 2014.

[11] D. Cox, J. Little, and D. O'Shea. *Using Algebraic Geometry*. Springer Verlag, 1998.

[12] W. Decker, G.-M. Greuel, G. Pfister, and H. Schönemann. *Singular* 3-1-6 — A computer algebra system for polynomial computations, 2012. http://www.singular.uni-kl.de.

[13] D.S. Dummit and R. Foote. *Abstract Algebra*. John Wiley and Sons, Inc., 2004.

[14] D. Eisenbud. *Commutative Algebra with a View Toward Algebraic Geometry*. Spinger, 2004.

[15] B. Grenet, P. Koiran, and N. Portier. On the complexity of the multivariate resultant. *Journal of Complexity*, 29:142–157, 2012.

[16] G-M Greuel and G. Pfister. *A Singular Introduction to Commutative Algebra*. Springer, 2008.

[17] R. Hartshorne. *Algebraic Geometry*. Springer Science Business Media, 1977.

[18] B. Hassett. *Introduction to Algebraic Geometry*. Cambridge University Press, 2007.

[19] J.D. Hauenstein and A. Jr. Liddel. Certified predictor-corrector tracking for newton homotopies. *Journal of Symbolic Computation*, 38:239–254, 2016.

[20] J.D. Hauenstein and F. Sottile. alphacertified: Certifying solution to polynomial systems. *ACM Transactions on Mathematical Software*, 74:28:1–28:19, 2012.

[21] R.A. Horn and C.R. Johnson. *Matrix Analysis*. Cambridge University Press, New York, 2010.

[22] S. Hosten and S. Sullivant. Ideals of adjacent minors. *Journal of Algebra*, 277:615–642, 2004.

[23] D. Kincaid and W. Cheney. *Numerical Analysis: Mathematics of Scientific Computing.* Brooks/Cole, Pacific Grove, 2002.

[24] D.G. Luenberger and Y. Ye. *Linear and Nonlinear Programming.* Spinger, New York, 2008.

[25] MATLAB. *8.5.0.197613 (R2015a).* The MathWorks Inc., Natick, Massachusetts, 2015.

[26] E.W. Mayr and A.R. Meyer. The complexity of the word problems for commutative semigroups and polynomial ideals. *Advances in Mathematics*, 46:305–329, 1982.

[27] S. McConnell. *Code Complete.* Microsoft Press, 2 edition, 2004.

[28] A. P. Morgan, A. J. Sommese, and C. W. Wampler. Computing singular solutions to polynomial systems. *Adv. in Appl. Math.*, 13(3):305–327, 1992.

[29] A. P. Morgan, A. J. Sommese, and C. W. Wampler. A power series method for computing singular solutions to nonlinear analytic systems. *Numer. Math.*, 63(3):391–409, 1992.

[30] A.P. Morgan and A.J Sommese. A homotopy for solving general polynomial systems that respects m-homogeneous structures. *Journal of Applied Mathematics and Computing*, 1987.

[31] M. Niemerg and D.J. Bates. Using monodromy to avoid high precision in homotopy continuation. *Mathematics in Computer Science*, 8(2):253–262, 2014.

[32] V. W. Noonburg. A neural network modeled by an adaptive Lotka-Volterra system. *SIAM Journal of Applied Mathematics*, 49:1779–1792, 1989.

[33] B. Stroustrup. *The C++ Programming Language.* Addison-Wesley, 2013.

[34] J. Verschelde. Phcpack: a general-purpose solver for polynomial systems by homotopy continuation. *ACM Transactions on Mathematical Software (TOMS)*, 25:251–276, 1999.

[35] J.H. Wilkinson. *Rounding errors in algebraic processes.* Dover, 1994. Reprint of the 1963 original [Prentice-Hall, Englewood Cliffs, N.J.].

Chapter 5

APPENDICES

The following appendices are describing two important input files used in the auto tuning section.

5.1. APPENDIX A

5.1.1. ILLUSTRATIVE EXAMPLE INPUT FILE.

```
INPUT


variable_group x,y,z;


function f1,f2,f3;


f1 = (y - x^2)*(x^2 + y^2 + z^2 - 1)*(x - 2);

f2 = (z - x^3)*(x^2 + y^2 + z^2 - 1)*(y - 2);

f3 = (z - x^3)*(y - x^2)*(x^2 + y^2 + z^2 - 1)*(z - 2);


END;
```

5.1.2. LAGRANGE POINTS INPUT FILE.

```
% LagrangePoints.input

% Equilibria for a 3rd small body rotating with two large

%      ones in circular orbit

INPUT

  function  fma1,fma2,dist13,dist23,fma3x,fma3y;

  % definition: w = omega^2 d12^3/(G m2)

  % The remaining variables are nondimensionalized as

  %   ratio to d12.

  variable_group w;

  variable_group r1;

  variable_group x,y,d13,d23;

  constant mu;

  % choose value for the mass ratio

  mu = 9;

  % the following eliminates r2

  r2 = 1-r1;

  % f=ma on mass 1

  fma1 = w*r1 - 1;

  % f=ma on mass 2

  fma2 = w*r2 - mu;

  % distance m1 to m3

  dist13 = (x-r1)^2 + y^2 - d13^2;

  % distance m2 to m3
  dist23 = (x+r2)^2 + y^2 - d23^2;

  % f=ma on m3

  a = w*d13^3*d23^3; b1 = mu*d23^3; b2 = d13^3;

  fma3x = a*x + b1*(r1-x) + b2*(-r2-x);

  fma3y = a*y + b1*(-y)   + b2*(-y);

END;
```