HEURISTIC PATH CHOICE TO AVOID ILL-CONDITIONING IN HOMOTOPY CONTINUATION

DANIEL J. BATES[1] AND TIMOTHY E. HODGES[1]

Abstract. Homotopy continuation is a numerical method rooted in numerical linear algebra. When paired with some theory from algebraic geometry, it provides a means for approximating solutions of polynomial systems of equations. The linear algebra at the core of homotopy continuation involves solving linear systems of equations built from the Jacobian matrix of the polynomial system. Ill-conditioning of the Jacobian matrix thus causes either a dangerous loss of accuracy or a significant computational cost penalty from using adaptive precision methods. This article introduces a novel method for detecting some zones of ill-conditioning and for building piecewise-linear homotopy paths that avoid these ill-conditioned zones.

Introduction

Homotopy continuation is a technique for approximating solutions of a system $\mathbf{F}(\mathbf{z};\mathbf{p})$ 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 $\mathbf{z_0}^{(\ell)}$, $\ell=1,\ldots,m$, 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 briefly in §1.1 and in great detail in [Allgower and Georg, 2003].

Much is gained by specializing to the case of systems of *polynomial* equations. For example, the number of solutions of $\mathbf{F}(\mathbf{z};\mathbf{p})=0$ is the same, say m, for almost every point $\mathbf{p}\in\mathbb{C}^k$. In fact, the subset $\mathcal{B}$ of parameter space at which the number of distinct solutions is less than m has measure zero. Points of $\mathcal{B}$ are referred to as *branch points*. Given projection $\pi:\mathbb{C}^N\times\mathbb{C}^k\to\mathbb{C}^k$ onto parameter space, points in the fiber over branch points at which two or more solution paths coincide are referred to as *ramification points*. The result, described in §1.1, is a probability one guarantee that a random path η through parameter space $\mathbb{C}^k$ will miss $\mathcal{B}$, i.e., solution paths will almost surely never intersect in the fiber of the projection π. This is fortunate, as the Jacobian matrix J (consisting of all first partial derivatives of the functions of $\mathbf{F}(\mathbf{z};\mathbf{p})$) is singular at ramification points but nonsingular everywhere else. Thus, the numerical linear algebra needed for tracking solution paths breaks down at branch points, causing tracking to fail.

While this probability one guarantee that one will never encounter a singular Jacobian matrix along a solution path sounds good, the reality is unfortunately not so pleasant. Recall that the *condition number*, $\kappa(M)$, of a matrix M is defined to be either the largest singular value of M divided by the smallest (if the smallest is nonzero) or ∞ (if not). Also, the condition number of the Jacobian matrix J of $\mathbf{F}(\mathbf{z};\mathbf{p})$ along a continuous path $\eta\subset\mathbb{C}^k\setminus\mathcal{B}$ is continuous. Thus, there is a region around each branch point at which J is near-singular,

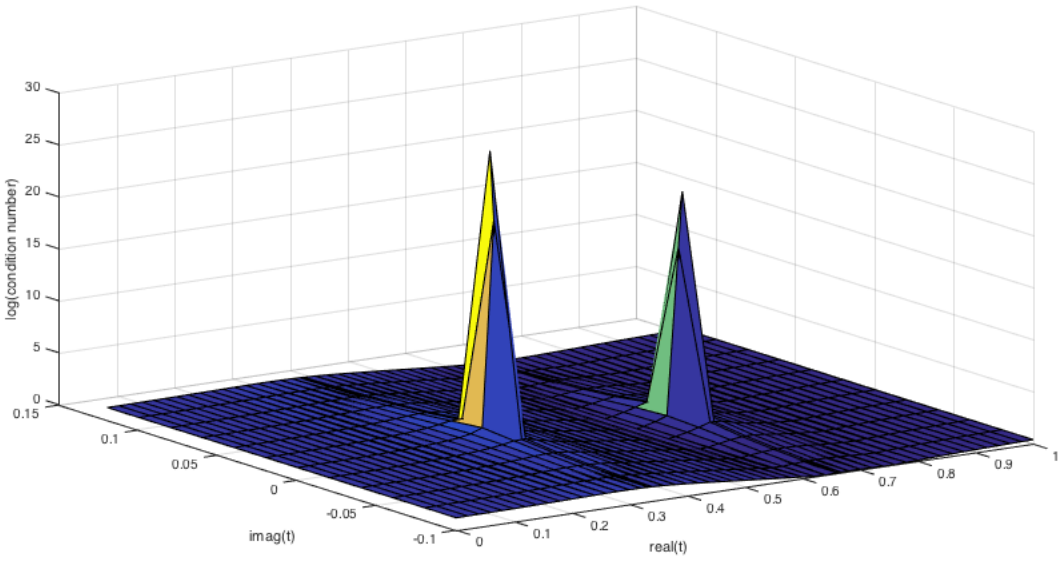

FIGURE 1. Plot of log of condition number of the Jacobian matrix for simple example (below) over a box in parameter (t) space, with ramification points at $t = \frac{1}{3}$ and $t = \frac{2}{3}$.

adversely affecting the accuracy of numerical linear algebra methods involving J. Indeed, a general rule [Wilkinson, 1994] is that the number of digits of accuracy (**Acc**) when solving a linear system with matrix J is approximately the precision being used to represent floating point numbers (**Prec**) minus the log of the condition number of J, i.e.,

$$\mathbf{Acc} \approx \mathbf{Prec} - \log(\kappa(J)).$$

These troublesome regions around branch points are referred to as *ill-conditioned zones*, and it is worth noting that these can show up away from branch points as well. Figure 1 shows the changing condition number of a homotopy with ramification points at $t = \frac{1}{3}$ and $t = \frac{2}{3}$.

In this article, we particularly consider the setting of *homotopy continuation*. This technique is the core computational method of most techniques in the field of *numerical algebraic geometry*. 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 that 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.

The purpose of this article is to introduce a novel method to handle ill-conditioned zones. The basic idea is to choose a piecewise linear path η one segment at a time, depending on the behavior of the solution paths being tracked along the previous segment of η. The key innovations of this article are the creation of such a path η instead of a standard straight line segment through parameter space and the use of fast local optimization methods to aid in the creation of η. While this method shows promise in theory and for small examples, there remains much room for optimization of the method, e.g., implementation choices; see Section 4 for details.

There has already been some work on handling ill-conditioning during homotopy continuation. Since accuracy depends on precision and the condition number, adaptive precision methods [Bates et al., 2008, 2009] that monitor the condition number and increase precision as necessary have been developed. While this method is the standard operating procedure of at least one software package, adaptive precision arithmetic is inherently expensive (at least as implemented in Bertini [Bates et al.]) and the monitoring of the condition number of the Jacobian matrix is wasted overhead away from ill-conditioned zones. Other methods have been developed to handle ill-conditioning at $t = 0$. These are known as endgames, and there are now a few standard options [Bates et al., 2011b, Morgan et al., 1992a,b]. In this article, we focus on the complement of that region, i.e., handling ill-conditioning away from $t = 0$. Finally, a method relying on monodromy was also recently proposed [Bates and Niemerg, 2014]. That method has a goal similar to that of this article – discovering ill-conditioned zones and handling them without adaptive precision – but the techniques are quite different in nature. In particular, the monodromy method passively relies on spikes in the condition number of the Jacobian matrix as a signal for ill-conditioned zones whereas the new method actively seeks out nearby zones using local optimization. Also, the monodromy method uses heuristically-chosen semicircles around ill-conditioned zones to avoid the use of higher precision while the new method navigates through the minefield of branch points on the fly, building a piecewise linear path.

The following section provides background on the necessary parts of homotopy continuation, numerical linear algebra, and monodromy. Nonlinear constrained optimization is necessary for this method, but knowledge of the basics of such techniques can be found elsewhere. Section 2 then walks through a running example of our technique, followed by the formal statement of the procedure. Section 3 provides details of a few other examples. This is followed by a final section that includes a discussion of the benefits and limitations of this method, as well as a number of open mathematical and implementation-specific problems (§4).

1. Background

In this section, we recount some foundational concepts from homotopy continuation, numerical linear algebra, and algebraic geometry needed for this article and not already described in the Introduction.

1.1. Homotopy continuation.

Let $\mathbf{f} : \mathbb{C}^N \to \mathbb{C}^N$ be a system of N polynomial equations in N variables.[1] We seek to find numerical approximations of all *isolated solutions* of $\mathbf{f}(\mathbf{z})$, i.e., all $\mathbf{z} \in \mathbb{C}^N$ for which $\mathbf{f}(\mathbf{z}) = 0$ and $|\mathbf{z} - \mathbf{w}| > \epsilon$ for any other solution $\mathbf{w} \in \mathbb{C}^N$, for some $\epsilon > 0$.

To compute the solutions of $\mathbf{f}(\mathbf{z}) = 0$, we define a homotopy, $\mathbf{H}(\mathbf{z}; t) : \mathbb{C}^N \times \mathbb{C} \to \mathbb{C}^N$, such that $\mathbf{H}(\mathbf{z}; 0) = \mathbf{f}(\mathbf{z})$ and $\mathbf{H}(\mathbf{z}; 1) = \mathbf{g}(\mathbf{z})$ is a system for which we know (or can easily compute) the solutions. There are many well-known methods for choosing $\mathbf{g}(\mathbf{z})$, collected and compared in Chapter 8 of [Sommese and Wampler, 2005]. The canonical choice of homotopy function is the *straight-line homotopy*,

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

[1] There are well-known ways of handling $N \neq n$ [Bates et al., 2015], so we restrict to the square case for simplicity.

By virtue of the construction of $\mathbf{H}(\mathbf{z};t)$, 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 for almost all choices of $t \in \mathbb{C}$. We refer to this number as ℓ. More precisely, the set $\mathcal{B} \subset \mathbb{C}$ consisting of points with fewer than ℓ solutions is an *algebraic set* (the solution set of a polynomial system) and is therefore of lower dimension than the one-dimensional ambient space $\mathbb{C}$. Hence, $\dim(\mathcal{B}) = 0$ and there are only finitely many points in t 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*. One method to achieve this randomness in η is the *gamma trick* [Li et al., 1989, Morgan and Sommese, 1989] where a random $\gamma \in \mathbb{C}$ is multiplied into the second term of homotopy $\mathbf{H}(\mathbf{z};t)$ in (1). The ℓ solutions above each point of η form one-real-dimensional curves – *solution curves* – that one may try to follow from solutions of start system $\mathbf{g}(\mathbf{z})$ at $t = 1$ to solutions of target system $\mathbf{f}(\mathbf{z})$ at $t = 0$.

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 2. Such computations involve only numerical linear algebra, i.e., solving some 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 $\mathbf{z}$.

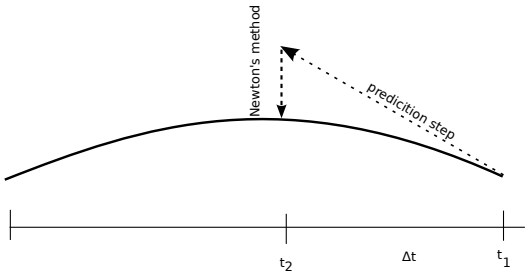

FIGURE 2. One step in the predictor-corrector scheme.

Finally, for the purposes of this paper, it is adequate to know that basic continuation is replaced with more sophisticated procedures called *endgames* [Bates et al., 2011b, Morgan et al., 1992a,b] at some pre-specified value of t, e.g., $t = 0.1$. The purpose of this switch is to extrapolate to $t = 0$ instead of walking directly to $t = 0$. While there is a probability one guarantee that η will have no branch points for $t \in (0, 1]$, there can be no such guarantee about target system $f(z)$ at $t = 0$.

There is much more to continuation that this brief overview, including the use of adaptive step sizes, adaptive precision [Bates et al., 2008, 2009], various choices of predictors [Bates et al., 2011a], and much more. The books [Bates et al., 2015, Sommese and Wampler, 2005] provide many more details.

1.2. **Numerical linear algebra.** Given matrix $\mathbf{A} \in \mathbb{C}^{n\times n}$ and vector $\mathbf{b} \in \mathbb{C}^n$, suppose we wish to approximate the solution $x \in \mathbb{C}^n$ of $\mathbf{Ax} = \mathbf{b}$ so that the computed solution $\hat{\mathbf{x}}$

satisfies $|\mathbf{x} - \hat{\mathbf{x}}| < \delta$ for some specified tolerance $\delta \in \mathbb{R}^+$ and choice of norm. Working with **Prec** digits of precision, a well-known result of Wilkinson [Wilkinson, 1994] gives us that the number of accurate digits of $\hat{x}$ is approximately

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

where $\kappa(\mathbf{A})$ is the condition number of $\mathbf{A}$. Thus, to retain a desired accuracy during continuation, one may actively adapt precision $\mathbf{P}$ during each predictor-corrector step, based on approximations of $\kappa(\mathbf{J})$ [Bates et al., 2008, 2009].

Unfortunately, the cost of using precision higher than that supported by hardware far outweighs the cost of using lower precision. For example, in the Bertini software package [Bates et al.], the base data type is `double` whereas higher precision is supported by use of the library MPFR [Fousse et al., 2007]. As reported in [Bates et al., 2008], the cost of going from double precision to virtually any level of higher precision is an increase in run time by a factor of at least 13. Thus, while adaptive precision is one way to address the issue of ill-conditioning, alternate methods could be favorable.

1.3. **Monodromy.** A natural idea to avoid ill-conditioned zones is to choose a path η *a priori* and simply veer off the path temporarily if J becomes ill-conditioned. This was suggested, for example, in [Kalaba and Tesfatsion, 1991]. After all, a branch point corresponds to the intersection of only a few branches (typically only two), so why should path $\eta \subset \mathbb{C}$ be chosen to work for *all* solution curves? Why not simply choose a different η for each solution curve?

The difficulty in doing so comes from *monodromy*. For example, consider the function

$$f(z,t) = z^2 - t,$$

where $t \in \mathbb{C}$. There is a branch point at $t = 0$ but two solutions ($\ell = 2$) for all other values of $t \in \mathbb{C}$. Beginning with $z = 1$ at $t = 1$, moving t around the unit circle once brings us to $z = -1$ in the fiber over $t = 1$. Repeating the trip around the unit circle brings us back to $z = 1$. This is the simplest example of monodromy, but more sophisticated examples are similar in nature.

For the purposes of this article, the key point is that choosing one path on each side of a branch point for distinct solution curves can cause those two solution curves to meet up on the other side of the branch point, resulting in the double tracking of one solution curve and the neglect of the other. In the example above, imagine tracking the solutions of $z^2 - t = 0$ from $t = 1$ to $t = -1$. Following one solution along the semicircle through $t = i$ and the other along the semicircle through $t = -i$ results in both curves arriving at the *same* solution at $t = -1$. Thus, for the sake of finding *all* solutions in homotopy continuation, it is necessary that all solution curves are tracked along the same path η in parameter space $\mathbb{C}$.

Coincidentally, monodromy has been used effectively in some numerical algebraic geometry techniques. For example, it is the fundamental concept underlying the *Cauchy endgame* [Morgan et al., 1992b]. It is also used in computing the numerical irreducible decomposition [Sommese et al., 2001]. Most recently, monodromy is at the source of another method for trying to avoid ill-conditioning [Bates and Niemerg, 2014].

2. Procedure

We begin this section by illustrating our method with a small example. We then give a formal statement of the method and related remarks.

2.1. **A simple example.** As a running example, consider the homotopy function

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

with variables x, y and parameter t. This system first appeared in another article aiming to find and avoid branch points [Bates and Niemerg, 2014]. There are six distinct solutions at almost all values of $t \in \mathbb{C}$, though there are at least two branch points. In particular, there are two triple roots at $t = \frac{2}{3}$ and three double roots at $t = \frac{1}{3}$. In practice, we are unaware of these branch points *a priori*.

Our mission 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.[2] To do so, we 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 and their ill-conditioned zones.

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 3. 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 propose to shift the solutions at $t = 1$ by a user-defined factor γ 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 $\mathbf{H}(x,y;t)$ at $t = 1$ should be near the solutions at $t = t'$.

Finding branch points in rectangle R. We use local optimization methods with objective function $c(x,y;t) = h_1^2 + h_2^2 + j_1^2 + j_2^2$ (more generally, $c(\mathbf{z};t) = h_1^2 + \cdots + h_n^2 + j_1^2 + \cdots + j_n^2$) to search for branch points within box R, where h_i are the homotopy functions and j_i are functions that will capture the rank-dropping of the Jacobian matrix of $\mathbf{H}(x,y;t)$ (more generally, $\mathbf{H}(\mathbf{z};t)$). There are various options for the functions j_i; the functions we employed are described thoroughly in [Bates et al., 2010]. We constrain our optimization problem to the interior of R; minimizers of $c(x,y;t)$ outside R are irrelevant in this step.

Notice that this optimization step could produce false positives. Indeed, a minimum $(x^*, y^*; t^*)$ of objective function $c(x,y;t)$ need not satisfy $\mathbf{H}(x^*, y^*; t^*) \approx 0$. 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] We seek to reach $t = 0.1$, not $t = 0$, as endgames [Bates et al., 2011b, Morgan et al., 1992a,b] begin operating at $t = 0.1$ or some other nonzero value of t.

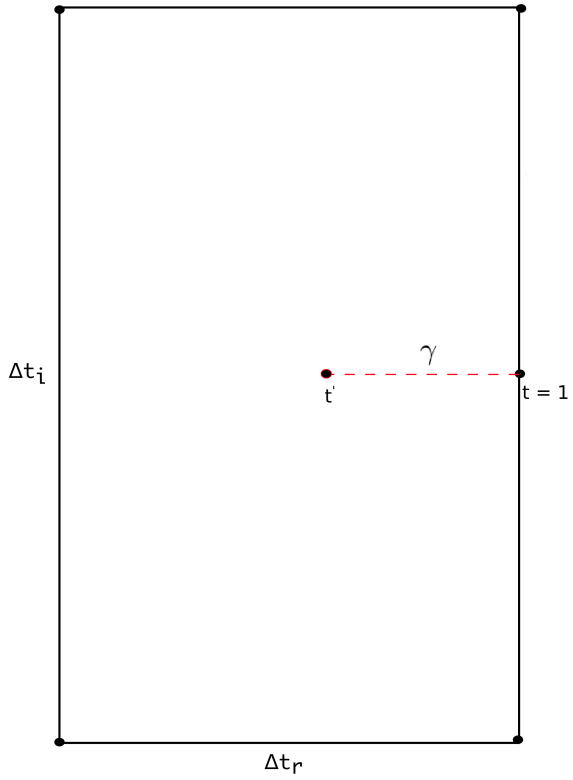

FIGURE 3. Rectangle in which branch points are sought.

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. Figure 4 shows two such guards.

One heuristic for creating the line segment of η from w_1 to w_2 depends on the angles between radially consecutive branch points. This can be seen in figure 5. After computing all angles between radially consecutive branch points, we simply bisect the largest angle, and choose w_2 in that direction. Since we are unaware of any branch points to the left of the left side of R, we choose w_2 some fraction μ of the distance to the left side of R.

In the case that the optimization phase produced no approximate branch points, w_2 should be chosen in the direction of $t = 0.1$, the ultimate target of path η.

Moving from $t = w_1$ to $t = w_2$. Once waypoint w_2 is chosen, the known solutions at w_1 may be tracked to w_2 using predictor-corrector methods. As mentioned above, adaptive precision should be used as not all ill-conditioned zones were necessarily discovered during the optimization phase of the method.

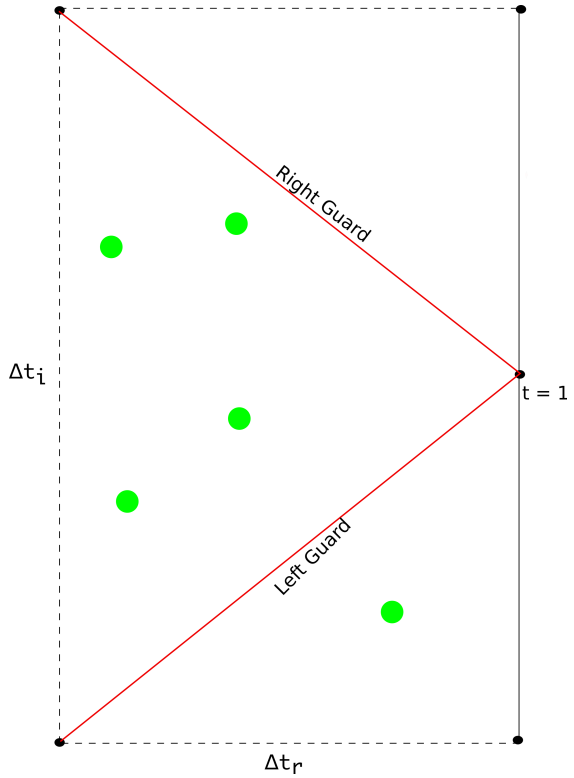

FIGURE 4. Rectangle R after optimization with minima (green dots) and guards (red line segments).

Further steps. Starting from waypoint w_i, the procedure to choose waypoint w_{i+1} is similar to that described for w_2 above. We create a box, search for branch points, and choose waypoint w_{i+1} so that line segment of η misses any discovered branch points.

The only significant difference between the first step and later steps is in the choice of direction. After all, it is easy to imagine a case in which η is driven away from the eventual target of $t = 0.1$ by potential branch points, e.g., if there are many branch points in the bottom half of each box but none in the top. In our limited experience, there have always been few enough branch points that the method always ends at $t = 0.1$ without intervention. It is not difficult to imagine a variant on this method that progressively favors the direction of $t = 0.1$ over the direction that optimally avoids all approximated branch points.

2.2. **Pseudocode and details.** In this section, we provide a formal statement of our method in pseudocode, followed by remarks about a few details.

Remarks.

(1) 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 time but may miss branch points; a slow, careful search takes more time

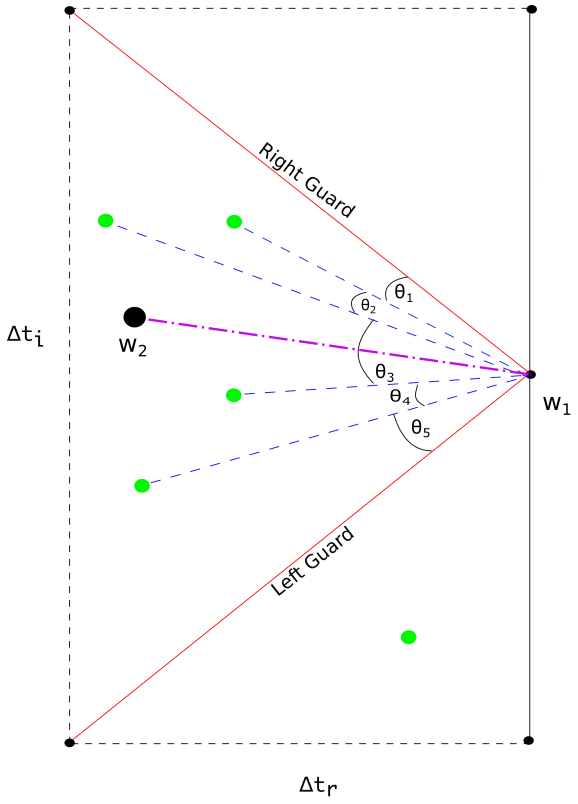

FIGURE 5. Choosing a path to waypoint w_2

but will also catch more branch points. Finding the happy medium remains an open problem.

(2) 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.

(3) 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.

2.3. **Our proof-of-concept implementation.** We implemented this heuristic using a combination of MATLAB and Bertini [Bates et al.]. To interface between these, we used the MATLAB package BertiniLab [Bates et al., 2016]. 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 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.

`Heuristic to determine path` η

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

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

Set $w_1 = 1, i = 1$.
Set $\Delta t_r = 0.1, \Delta t_i = 0.2, \gamma = 0.5$ if not provided.
while $t \neq 0.1$ **do**
    Set $\mathcal{B} = \emptyset$.
    Set $t' = w_i - \gamma \Delta t_r$.
    Set $b = 0$. ▷ Counts branch points discovered.
    for $\ell = 1, \ldots, m$ **do**
        Run optimization problem

$$\begin{array}{ll} \underset{\mathbf{z};t}{\text{minimize}} & c(\mathbf{z};t) \\ \text{subject to} & t \in R, \end{array}$$

        starting from ℓ^{th} point of S. ▷ Box R constructed from $w_i, \Delta t_r, \Delta t_i$
        if (Convergence to a branch point within R) **then**
            Store minimum in $\mathcal{B}$, increment b.
        end if ▷ Ignore runs that leave R.
    end for
    Order $\mathcal{B}$ based on imaginary part.
    for $j = 1, \ldots, b-1$ **do**
        Find angle θ_j between $\mathcal{B}_j$ and $\mathcal{B}_{j+1}$, measuring from w_k.
    end for
    Also compute angles θ_0 between left guard and $\mathcal{B}_1$ and θ_b between $\mathcal{B}_j$ and right guard.
    Find j^* such that $\theta_{j^*} > \theta_j \forall j \neq j^*$. ▷ Max angle between branch points.
    Let $\mathbf{v}_{j^*}$ be the vector that makes the angle θ_{j^*} with the horizontal.
    Increment i.
    Set $w_i = w_{i-1} + \frac{\mu}{cos(\theta_{j^*})}\mathbf{v}_{j^*}$
    ▷ If $\mathcal{B} = \emptyset$ then $\mathbf{v}_{j^*}$ is in direction of the vector from w_{i-1} to 0.1.
end while

3. Examples

3.1. **Neural Network Example.** Our second example comes from neural networks studied by Noonburg [Noonburg, 1989]. The system from that paper is embedded in the following total degree homotopy:

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

Figure 6 shows the path η that was constructed with our heuristic. Notice that there appears to be a cluster of potential ramification points near $t = \frac{1}{2}$. In fact, there is a special branch point at $t = \frac{1}{2}$, where 10 pairs of paths come together. This causes minima to be detected in several boxes around $t = \frac{1}{2}$ since higher multiplicity ramification points seem to yield larger ill-conditioned zones. A careful analysis of this phenomenon would be interesting but is beyond the scope of this article.

It is perhaps interesting to note that a computation to find *all* branch points for this system yielded 42 solutions, meaning there could be as many as 42 branch points. This was accomplished by solving the original system, augmented by the determinant of the Jacobian of the system. Since the $t = \frac{1}{2}$ branch point accounts for 20 of these, there are no more than 22 other branch points around the complex plane.

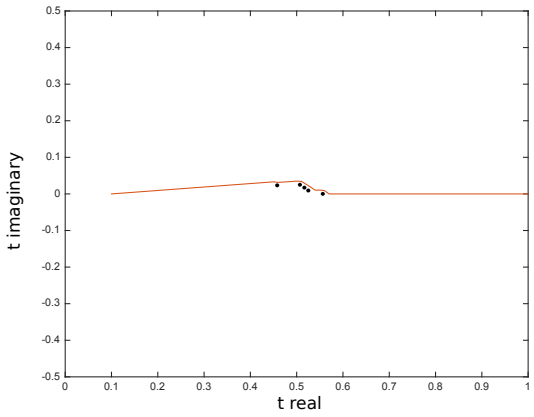

FIGURE 6. Path η determined by one run of our heuristic on the neural network example.

3.2. **A family of larger examples.** Another set of examples is the family of adjacent minor systems. Consider a matrix of size $2 \times n$ filled with indeterminants, as shown below.

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

We consider the 2×2 adjacent minors of M, using only columns adjacent to each other. This yields $n-1$ equations in the $2n$ variables $\mathbf{x} = (x_1, x_2, \dots, x_{2n})$. These equations are

$$\begin{aligned} f_1 &= x_1x_{n+2} - x_2x_{n+1} \\ f_2 &= x_2x_{n+3} - x_3x_{n+2} \\ f_3 &= x_3x_{n+4} - x_4x_{n+3} \\ &\vdots \\ f_{n-1} &= x_{n-1}x_{2n} - x_nx_{2n-1} \end{aligned}$$

This system has been studied extensively and is a complete intersection with codimension is $n-1$ and degree is 2^{n-1} [Hosten and Sullivant, 2004]. We may add linear equations with random complex coefficients to make this system square. This also reduces the solution set to isolated solutions. These linear equations are denoted $L_1, L_2, ..., L_{n+1}$.

We construct the following homotopy:

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

The output of two runs of our heuristic method for $n = 3$ are shown in Figure 7. Both runs indicate an ill-conditioned zone near $t = 0.8$, though the second run encounters a possible ill-conditioned zone that was not detected during the first run.

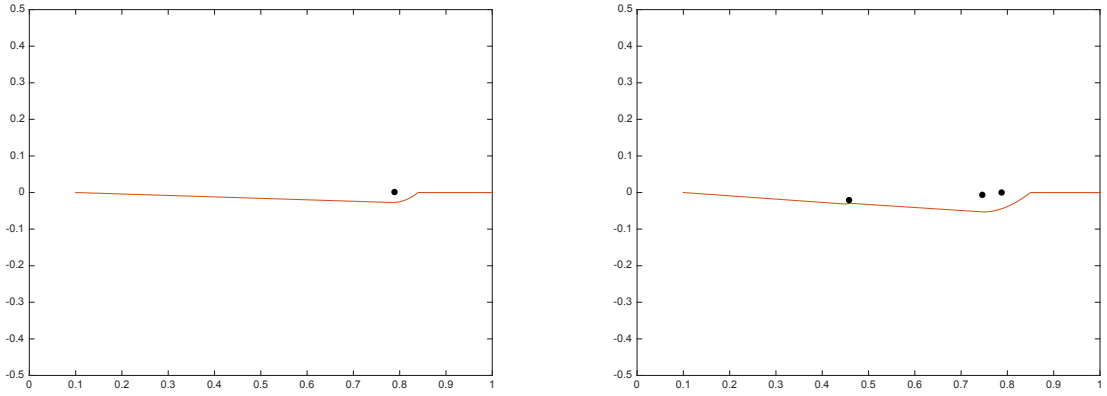

FIGURE 7. Two runs of our heuristic on the 2×3 adjacent minor system.

4. Discussion

4.1. **Benefits and limitations.**

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

Timing and scaling. Figure 8 shows 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 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 meaningless; this new method, as currently implemented, cannot compete.

Problem	Total 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 8. Average run times for running our proof-of-concept implementation on five examples several times each, in seconds.

4.2. **Alternate methods.** There are currently four methods for handling branch points during homotopy continuation related to this method: Using adaptive precision (the 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 [Bates and Niemerg, 2014]. This article 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 $\mathbf{H}(\mathbf{z}, t)$ [Bates et al., 2010]. 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 [Bates and Niemerg, 2014], 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 [Bates and Niemerg, 2014] are both heuristics, with various benefits and limitations, and the optimal method might be some combination of the two (or neither). A decision on this can be made only after a careful implementation of all options has been completed.

4.3. **Open problems.** This new method is a theme with many possible variations. In this section, we briefly record some options to be considered once a careful implementation has been completed.

(1) **Ill-conditioned zone geography.** The method currently seeks to stay as far from the approximated branch points as possible. Of course, staying as near the real line as possible could result in a much shorter path η, thereby reducing run times. This also plays into the choice of μ. How far must we stay from branch points to avoid any loss of accuracy? What is the actual shape of an ill-conditioned zone? These are difficult questions to answer, but these answers are important as we seek to improve such methods.
(2) **Box dimensions.** Very small boxes can each be searched rapidly, but the result is a large number of boxes between $t = 1$ and $t = 0.1$, meaning that the tracker creeps

slowly along a polygonal path η with many short segments. Large boxes require more search time (or more negligence during the search), but there are then fewer boxes to reach $t = 0.1$. What is optimal? Perhaps an adaptive box size method could be developed, depending on the behavior of the method on previous boxes, as with adaptive step size.

(3) **Endgame zone recognition.** Our heuristic provides a means to try to locate branch points. The specialized methods known as endgames (§1) are only successful once the path has been tracked beyond the branch point nearest $t = 0$. The method of this article could be adapted to sweep around $t = 0$, for example in a Cauchy endgame loop, to search for branch points near $t = 0$.

(4) **Piecewise nonlinear paths.** The path between consecutive waypoints need not be linear. Tracking difficulties aside, it is reasonable to expect that it will sometimes be easier to thread a nonlinear path (or perhaps a polygonal path) between branch points than a single linear path.

(5) **Solution-specific paths.** In this method, we choose one homotopy path η along which we follow all solution paths. It is tempting to build a path η specific to each solution. After all, it is reasonable to believe that almost all ramification points are multiplicity two singularities (where only two solution curves come together), so why should η shift for *all* solutions instead of just those two? The risk of solution-specific paths lies with monodromy, as described in §1. Some careful book-keeping and rerunning of paths that collided could mitigate this difficulty, but fleshing out such a method is beyond the scope of this article.

(6) **Other variants.** This method relies on many tolerances, and the behavior of the method will vary as these tolerances are changed. There is much left to study in this regard.

4.4. **Acknowledgments.** The authors were partially supported by NSF grants DMS–1115668 and ACI–1440467. The first author would also like to thank Andrew Sommese and Charles Wampler for stimulating conversations about the synchronization of paths. While this article is not a direct consequence of our conversations, there is clearly a connection between the two ideas. We would also like to thank Anna Seigal for her valuable suggestions for variations on this method.

References

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

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

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

D.J. Bates, J.D. Hauenstein, A.J. Sommese, and C.W. Wampler. Adaptive multiprecision path tracking. *SIAM J. Num. An.*, 46:722–746, 2008.

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.

D.J. Bates, J.D. Hauenstein, C. Peterson, and A.J. Sommese. Numerical decomposition of the rank-deficiency set of a matrix of multivariate polynomials. pages 55–77, 2010. doi: 10.1007/978-3-211-99314-9_2. URL http://dx.doi.org/10.1007/978-3-211-99314-9_2.

D.J. Bates, J.D. Hauenstein, and A.J. Sommese. Efficient path tracking methods. *Numer. Algor.*, 58:451–459, 2011a.

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

D.J. Bates, J.D. Hauenstein, A.J. Sommese, and C.W. Wampler. *Bertini: Software for numerical algebraic geometry.* 2015.

D.J. Bates, A.J. Newell, and M. Niemerg. Bertinilab: A matlab interface for solving systems of polynomial equations. *Numer. Algorithms*, 71(1):229–244, January 2016. ISSN 1017-1398. doi: 10.1007/s11075-015-0014-6. URL http://dx.doi.org/10.1007/s11075-015-0014-6.

L. Fousse, G. Hanrot, V. Lefèvre, P. Pélissier, and P. Zimmermann. Mpfr: A multiple-precision binary floating-point library with correct rounding. *ACM Trans. Math. Softw.*, 33(2), June 2007. ISSN 0098-3500. doi: 10.1145/1236463.1236468. URL http://doi.acm.org/10.1145/1236463.1236468.

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

R. Kalaba and L. Tesfatsion. Solving nonlinear equations by adaptive homotopy continuation. *Applied Mathematics and Computation*, 41(2):99 – 115, 1991.

T. Y. Li, T. Sauer, and J. A. Yorke. The cheater's homotopy: An efficient procedure for solving systems of polynomial equations. *SIAM J. Numer. Anal.*, 26(5):1241–1251, October 1989. ISSN 0036-1429. doi: 10.1137/0726069. URL http://dx.doi.org/10.1137/0726069.

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, 1992a.

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

A.P. Morgan and A.J. Sommese. Coefficient-parameter polynomial continuation. *Appl. Math. Comput.*, 29(2):123–160, January 1989. ISSN 0096-3003. doi: 10.1016/0096-3003(89)90099-4. URL http://dx.doi.org/10.1016/0096-3003(89)90099-4.

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

A.J. Sommese and C.W. Wampler. *Numerical solution of polynomial systems arising in engineering and science.* World Scientific, Singapore, 2005. ISBN 3-540-19468-1.

A.J. Sommese, J. Verschelde, and C.W. Wampler. Numerical decomposition of the solution sets of polynomial systems into irreducible components. *SIAM J. Numer. Anal.*, 38(6): 2022–2046, 2001.

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

Department of Mathematics, Colorado State University, Fort Collins, CO 80523 (USA)
E-mail address: bates@math.colostate.edu

Department of Mathematics, Colorado State University, Fort Collins, CO 80523 (USA)
E-mail address: hodges@math.colostate.edu