

# Efficient endgames and path tracking methods

Daniel J. Bates <sup>\*</sup>      Jonathan D. Hauenstein <sup>†</sup>

Andrew J. Sommese <sup>‡</sup>

September 8, 2009

## Abstract

Path tracking is the fundamental computational tool in homotopy continuation and is therefore key in most algorithms in the emerging field of numerical algebraic geometry. Though the basic notions of predictor-corrector methods have been known for years, there is still much to be considered, particularly in the specialized algebraic setting of solving polynomial systems. This article considers two ways in which path tracking can be made more efficient. First, a novel parallel algorithm for performing endgames at the end of homotopy paths, based on the Cauchy endgame, is presented, along with some heuristics useful in its implementation. Second, the effects of the choice of predictor method on the performance of a tracker is analyzed, along with notes on how to employ Runge-Kutta methods in conjunction with adaptive precision. These ideas have been implemented in the Bertini software package with several examples provided.

**Keywords.** path tracking, homotopy continuation, endgames, parallelism, numerical algebraic geometry, polynomial systems, ordinary differential equations, Euler's method, Runge-Kutta methods.

**AMS Subject Classification.** 14D05, 65H10, 65H20, 65E05, 65L06, 65Y05

---

<sup>\*</sup>Department of Mathematics, Colorado State University, Fort Collins, CO 80523 (bates@math.colostate.edu, <http://www.math.colostate.edu/~bates>). This author was supported by the Institute for Mathematics and Its Applications (IMA).

<sup>†</sup>Fields Institute, Toronto, ON, Canada M5T 3J1 (jhauenst@fields.utoronto.ca, <http://www.fields.utoronto.ca/~jhauenst>). This author was supported by the Fields Institute, Duncan Chair of the University of Notre Dame; the University of Notre Dame Center for Applied Mathematics; and NSF grants DMS-0410047 and NSF DMS-0712910.

<sup>‡</sup>Department of Mathematics, University of Notre Dame, Notre Dame, IN 46556 (sommese@nd.edu, <http://www.nd.edu/~sommese>). This author was supported by the Duncan Chair of the University of Notre Dame; and NSF grants DMS-0410047 and NSF DMS-0712910.

# 1 Introduction

Let  $H(z, t)$  be a system of  $N$  polynomials with  $(z, t) \in \mathbb{C}^N \times \mathbb{C}$  such that given any element  $x^*$  of the finite set  $\mathcal{F}$  of nonsingular isolated solutions of  $H(z, 1) = 0$ , the connected component  $\mathcal{C}$  of the set

$$\{(z, t) \in \mathbb{C}^N \times (0, 1] \mid H(z, t) = 0\} \quad (1)$$

that contains  $x^*$  is the graph of a differentiable map  $t \rightarrow x(t)$  with  $x(1) = x^*$ . We say that  $H(z, t)$  is a good homotopy with respect to  $\mathcal{F}$  and that  $x(t)$  is the solution path starting at  $x^* \in \mathcal{F}$ .

Most of the effort in the numerical computation of the solution sets of systems of polynomials comes down to computing the limits as  $t$  goes to 0 for solution paths  $x(t)$  starting at the solution  $x^* \in \mathcal{F}$ .

For example, if

$$f(z) = \begin{bmatrix} f_1(z_1, \dots, z_N) \\ \vdots \\ f_N(z_1, \dots, z_N) \end{bmatrix} \quad (2)$$

is a system of polynomials on  $\mathbb{C}^N$ , there are numerous ways to set up homotopies  $H(z, t)$  with  $H(z, 0) = f(z)$  [24]. For such homotopies, the set of *startpoints*  $\mathcal{F}$  consists of all isolated nonsingular solutions of the *start system*  $H(z, 1) = 0$ . The set of *endpoints*, i.e., limits of the solution paths starting at points of  $\mathcal{F}$ , contains all isolated solutions of  $f(z) = 0$ , the *target system*. The special variable  $t$  is called the *path variable* or *parameter* for the homotopy. Although it is sometimes useful to allow for multiple parameters, we restrict to the case of a single parameter in this article. For more background, [1, 13, 14, 24] are good references for numerical homotopy methods.

If an isolated solution  $\hat{x}$  of  $H(z, 0) = 0$  is the limit of some number  $d \geq 1$  solution paths  $x(t)$  as  $t \rightarrow 0$ , then  $d$  is an upper bound on the *multiplicity*  $\mu$  of  $\hat{x}$  as a solution of  $H(z, 0) = 0$ . Singular isolated solutions of  $f(z) = 0$ , i.e., isolated solutions with multiplicity  $\mu \geq 2$ , are typically more expensive to compute than nonsingular solutions. This is because the linear algebra connected with path-tracking is ill-conditioned near such singular limit points, requiring the use of costly higher precision. Additionally, all of the paths leading to the same limit point will be followed since, *a priori*, there is no way to decide which paths will lead to the same solution.

To help mitigate the numerical difficulties encountered when trying to compute singular endpoints, a variety of techniques more sophisticated than standard path tracking have been developed [11, 16, 17, 18, 24]. One of these

is the Cauchy endgame [16]. As described in Section 2, this endgame employs a numerical version of Cauchy’s integral formula on loops in the parameter space around  $t = 0$  to produce approximations of the endpoint at  $t = 0$ .

In the first part of this article, we show that by using the Cauchy endgame, we can identify those paths which lead to the same limit point *without* tracking every path all the way to convergence at  $t = 0$ . Besides reducing the work in computing the same endpoint, this significantly decreases the amount of post processing needed to decide which endpoints are the same. Not only does the Cauchy endgame have the above compelling properties, but, unlike other endgames, it parallelizes well, as demonstrated below. We also present a pair of heuristics for deciding whether a given value of  $t$  is less than the modulus of all nonzero singular points in the parameter space (i.e., the nonzero ramification points). This is valuable information since Cauchy loops with images containing nonzero singular points are useless in computing approximations at  $t = 0$ . Skipping useless Cauchy loops (i.e, not beginning the endgame until all Cauchy loops are worth computing) clearly saves computation time.

Predictor/corrector path tracking methods make use of ordinary differential equation solvers for prediction and Newton’s method for corrections. In the second part of this article, we consider the consequences of various choices of a differential equations solver and explain how to incorporate adaptive precision methodologies into Runge-Kutta methods. Though one would expect higher order methods to be more efficient, unpublished work by other researchers in numerical algebraic geometry seemed to indicate that all methods are approximately equal in the algebraic predictor/corrector setting. The results in this article indicate that there is indeed value in using higher-order methods such as the fifth-order Runge-Kutta-Fehlberg method (RKF45). Though this fact should come as no surprise to those familiar with numerical methods for differential equations, it debunks inaccurate folklore in the numerical algebraic geometry community.

In §2, we recall the original Cauchy endgame. In §3, we introduce our more efficient Cauchy endgame algorithm both for a single processor and in parallelized form, and also provide the aforementioned heuristic for deciding when to begin using this endgame. In §4, we compare the performance of the power series endgame, the classical Cauchy endgame, and our new version of the track-back Cauchy endgame.

In §5, we recall Euler’s method as well as higher-order methods. In §6, we indicate how to adapt precision on the fly while using higher-order methods and also how error control is different with Runge-Kutta methods than with the standard Euler/Newton scheme. Finally, computational evidence

supporting the value of higher-order methods in this situation is provided in §7.

## 2 The Cauchy endgame

### 2.1 Background

Given a polynomial system  $f(z)$ , there are a number of ways to form a homotopy  $H(z, t) : \mathbb{C}^n \times \mathbb{C} \rightarrow \mathbb{C}^n$  with all of the desirable properties necessary to perform homotopy continuation. Several of these homotopy types are described in [24], as is homotopy continuation. For this article, it is enough to know that there are solution paths  $x(t)$  leading from the solutions of  $H(z, 1)$  to solutions of  $H(z, 0) = f(z)$  and that these paths may be followed using homotopy continuation (predictor/corrector methods). Standard numerical predictor/corrector methods encounter difficulty near singularities, requiring the use of (expensive) higher precision and shorter steplengths. Thus, it is of computational value to stay as far from singularities as possible. This is the role of *endgames* in homotopy continuation; these are specialized numerical methods intended to speed convergence of the approximation of the endpoint of  $x(t)$  as  $t \rightarrow 0$ .

The Cauchy endgame was first reported in [16] as one of several potential endgames for use with homotopy continuation (see also [24]). The idea is to use the Cauchy integral formula to approximate  $H(z, 0)$  using loops in the parameter space about 0 of radius  $t$ , for varying values of  $t$ . In this section, we describe the theory behind this numerical method; the next section contains a few implementation notes. Since most homotopies have  $\mathbb{C}$  as the parameter space, we will assume that our parameter space is  $\mathbb{C}$ .

Each isolated solution  $x^*$  of  $H(z, 0)$  is the endpoint of at least one solution path of the homotopy. In fact, there is some number of paths, say  $d$ , with endpoint  $x^*$ . For a small value of  $t$ , say  $t'$ , near enough 0, the path values of these  $d$  paths will be permuted by moving around the loop  $t'e^{i\theta}$  for  $\theta \in [0, 2\pi]$ . In fact, the corresponding permutation can be broken up into some number of cycles, say  $k$ . Let  $\gamma_i$  denote the number of elements in the  $i^{\text{th}}$  cycle, with  $\sum_{i=1}^k \gamma_i = d$ . The integers  $\gamma_1, \dots, \gamma_k$  are called the winding numbers of  $x^*$ .

More technically, given a nontrivial algebraic map  $p$  of a one-dimensional algebraic set  $X$  to  $\mathbb{C}$ , there is a finite set  $F \subset \mathbb{C}$  such that the restriction of  $p$  to  $X - p^{-1}(F)$  is a covering of  $\mathbb{C} - F$  with all fibers having the same number of points. See Appendix A of [24] for further details about the underlying theory.

Given a point  $a$  in an open subset  $U \subset \mathbb{C}$  and a disk  $D$  with  $a \in D \subset U$ , the Cauchy integral formula computes the function value  $f(a)$  as

$$f(a) = \frac{1}{2\pi i} \int_C \frac{f(z)}{z - a} dz, \quad (3)$$

where  $C$  is the boundary of disk  $D$  and the integral is taken to be a contour integral. A numerical integration method, e.g., the trapezoid method, may be used to approximate the integral.

In the homotopy setting, we compute approximations to a solution curve  $x(t)$  of  $H(z, t)$  for values of  $t$  from 1 to 0. As  $t$  approaches 0, starting at  $t = 0.1$ , for example, we may begin computing approximations of  $H(z, 0)$  using  $a = 0$ , the loop in the parameter space given by  $te^{i\theta}$  for  $\theta \in [0, 2\pi]$ , and a numerical method for evaluating the integral above. As  $t$  marches to 0, these approximations will converge, so we may terminate this endgame (and the path) once two consecutive approximations are within a prescribed tolerance.

## 2.2 Implementation

The Cauchy endgame has been implemented in Bertini [2]. This section describes the original implementation of this endgame in Bertini, though it has recently been improved using the ideas from §3.

Given a solution path  $x(t)$ , Bertini will first track the path to the endgame boundary,  $t'$ , set to 0.1 by default. At that point, Bertini sends the path variable  $t$  around the circle  $t'e^{i\theta}$  for  $\theta \in [0, 2\pi]$ . Since it is simpler to follow lines than curves in  $t$ , Bertini actually discretizes this circle into a polygon and tracks around this polygonal path repeatedly until the path value of  $x(t)$  at the end of a loop is very near the starting path value  $x(t')$  (within a prescribed tolerance), i.e., until the Cauchy loop closes.

Once the Cauchy loop closes, it is straightforward to use numerical integration techniques with the Cauchy integral formula to compute an approximation at  $t = 0$ . After this first approximation of  $x(0)$  has been computed,  $t'$  is reduced by some factor (typically  $\frac{1}{2}$ ), and the procedure is repeated. The endgame terminates once two consecutive approximations of  $x(0)$  agree to the desired tolerance.

### 3 A new Cauchy endgame

#### 3.1 Overview

The motivation for the revised Cauchy endgame is to recognize those paths which coalesce to some solution  $x^*$  of  $H(z, t) = 0$  at  $t = 0$  without wasting the time needed to follow all paths to  $t = 0$  and determine which endpoints are the same. Let  $t = t^*$  be the value of the path variable from which the approximation to  $t = 0$  finally converges for some given path. Let  $\gamma \leq \mu$  be the winding number of  $x^*$  for the given path.

At  $t = t^*$ , we will have tracked around  $t = 0$   $\gamma$  times, thereby collecting all  $\gamma$  solutions of  $H(z, t^*) = 0$  which lead to  $x^*$  for this cycle. Call this set of points  $\mathcal{G}$ . We may then follow the paths beginning at points of  $\mathcal{G}$  backwards to discover which of the solutions of  $H(z, 1) = 0$ , i.e., which starting points lead to  $x^*$ . Note that in doing so, we only run the endgame for one path of  $\mathcal{G}$ .

At first glance, it may seem that there is little savings in this method over the original Cauchy endgame. However, endgames are the expensive part of path-tracking. Thus, being able to avoid running the endgame for  $\gamma - 1$  of the  $\gamma$  paths in the same cycle leading to the same endpoint, for example, save a significant amount of computational time, particular if  $\gamma \gg 1$ . Of course, this back-tracking is unnecessary if  $\gamma = 1$ .

#### 3.2 Heuristics about when to start the endgame

The Cauchy endgame can be very inefficient when extraneous Cauchy loops are computed where convergence probably will not occur. In this section, we present a pair of heuristics that help to eliminate this wasted computational effort. After passing both heuristic tests, the standard Cauchy endgame is utilized with the goal of being in a region where convergence can be attained quickly.

Endgames are known to converge within an annulus around  $t = 0$  [24]. The Cauchy endgame requires that no singular point other than (possibly)  $t = 0$  falls in the interior of the image of the Cauchy loop. Thus, the outer loop of the annulus is determined by the (unknown) set of singular (ramification) points in the parameter space. The inner loop is determined by the preponderance of numerical error very near  $t = 0$  when using fixed precision. This annulus is called the *endgame operating zone*. Since only the outer loop matters when using adaptive precision, the radius of that loop is given a name, the *endgame convergence radius*. Note that the actual region from which the endgame may converge is a superset of the endgame

operating zone, but, in practice, it is reasonable to think of this region as an annulus.

Let  $\gamma \geq 1$  be the winding number for the endpoint of the path  $x(t) : \mathbb{C} \rightarrow \mathbb{C}^n$  at  $t = 0$ . In the endgame operating zone, it is known that  $x(t)$  has a Puiseux series expansion, namely  $x(t) = x(0) + \sum_{j=1}^{\infty} a_j t^{j/\gamma}$ . To avoid trivialities in this section, we shall assume that  $x(t)$  is nonconstant. Let  $c = \min\{j | a_j \neq 0\}$  and let  $v \in \mathbb{C}^n$  be a random vector. Then,

$$v \cdot x(t) = v \cdot x(0) + v \cdot a_c t^{c/\gamma} + \sum_{j=c+1}^{\infty} v \cdot a_j t^{j/\gamma} \quad (4)$$

with  $v \cdot a_c \neq 0$ .

The first heuristic method approximates the value of  $\frac{c}{\gamma}$  and tests for agreement between two such approximations. To approximate this value, three points along the path  $x(t)$  are collected in a geometric sequence, say  $x(R)$ ,  $x(\lambda R)$ , and  $x(\lambda^2 R)$ , for some  $0 < \lambda < 1$  and  $R > 0$ . Using Eq. 4,

$$g(R) := \frac{\log \left| \frac{v \cdot x(\lambda R) - v \cdot x(\lambda^2 R)}{v \cdot x(R) - v \cdot x(\lambda R)} \right|}{\log \lambda} \approx \frac{c}{\gamma}. \quad (5)$$

To pass the heuristic test,  $g(R)$  must be positive, and  $g(R)$  and  $g(\lambda R)$  must “agree.” For some  $0 < L < 1$ , agreement is defined as

$$L < \frac{g(R)}{g(\lambda R)} < \frac{1}{L}. \quad (6)$$

Bertini uses  $L = \frac{3}{4}$ , and, as a fail-safe mechanism to avoid the possibility of never passing the test due to numerical error, Bertini automatically moves on to the next heuristic test if it tracks to a value of  $t$  which is smaller than  $10^{-8}$ .

The second heuristic method compares values around the Cauchy loops. When a loop contains an erroneous branch point or the radius is too large for convergence, the values around the loop generally differ by large amounts. One way of enforcing that the values do not differ radically without running the whole Cauchy endgame is to collect sample points on  $x(Re^{i\theta})$ ,  $\theta \in [0, 2\pi]$  and determine if the minimum and maximum norms of these sample points, say  $m$  and  $M$ , respectively, are “well-behaved.” For  $\beta > 0$  and  $0 < K < 1$ , the  $m$  and  $M$  are well-behaved if either  $M - m < \beta$  or  $\frac{m}{M} > K$ . Bertini takes  $\beta$  to be the requested final tolerance of the endpoint and  $K = \frac{1}{2}$ . As in the first heuristic test, Bertini employs a fail-safe mechanism to automatically start the standard Cauchy endgame if it tracks to a value of  $t$  which is smaller than  $10^{-14}$ .

### 3.3 Parallel version

In the standard single-processor version of homotopy continuation, paths are tracked sequentially, i.e., one after another. One way to parallelize homotopy continuation is to send a packet of paths to each processor. Though this way of parallelizing homotopy continuation is straightforward, there is value in considering how to distribute the work among the available processors to minimize total running time.

In Bertini, the paths are dynamically distributed in packets with the size of the packets decreasing exponentially. That is, the size of the first packet is much larger than the size of the last packet. This provides for a more uniform load balance for the processors since not all paths take the same amount of time. Indeed, those ending at singular endpoints or passing near a singularity take considerably longer than those which stay well-conditioned throughout the entire path.

In the Bertini implementation of the new endgame described above, the manager maintains a list of startpoints for which the endpoint is unknown. When a worker process is available, the manager sends it a packet of startpoints, with the sizes of the packet decreasing exponentially, as before. The worker process sequentially computes the endpoint of the path for each of the start points it received and back-tracks when necessary. Before tracking each path, the startpoint is compared with the track-back points computed for this packet. After each endpoint is known for the startpoints in the packet, the data is sent back to the manager who updates the list of startpoints and sends another packet.

This parallelization can result in running the endgame more than using a sequential processing if the paths on the same cycle are simultaneously sent to different workers. The additional communication costs to avoid this would, in general, be more expensive than the cost of the extra computations. By reducing the maximum size of the packets, the likelihood of such an event occurring decreases, but this creates more communication between the manager and the workers. To maintain a good balance, we found that a maximum of 20 paths per packet works well.

## 4 Implementation details and computational results

The track-back Cauchy endgame is implemented in the software package Bertini [2]. All the examples discussed below were run on a 2.4 GHz Opteron

250 processor with 64-bit Linux. The parallel examples were run on a cluster consisting of a manager that uses one core of a Xeon 5410 processor and 8 computing nodes each containing two 2.33 GHz quad-core Xeon 5410 processors running 64-bit Linux, i.e., one manager and 64 workers.

In the examples presented, the paths were tracked using adaptive precision [3, 4]. The power series endgame collected sample points along the path at  $t = 4^{-k}$ ,  $k = 1, 2, \dots$ , and used four successive sample points to approximate the endpoint. Similarly, the Cauchy endgame computed approximations of the endpoint at  $t = 4^{-k}$ ,  $k = 1, 2, \dots$  using four sample points per loop. For both endgames, the stopping criterion was having two successive approximations agree to a tolerance of  $10^{-10}$ .

#### 4.1 Solutions at infinity

Solutions at infinity tend to have large winding numbers, which leads to computational difficulty when trying to compute the endpoints accurately. Solving the first stage of the cascade algorithm [21, 24] is one place where solutions at infinity waste computational resources.

For example, on  $\mathbb{C}[x, y, z]$ , let  $B$  be a random  $3 \times 3$  unitary matrix over  $\mathbb{C}$ ,  $L_1, L_2$ , and  $L_3$  be general linear functions,

$$L = \begin{bmatrix} L_1 \\ L_2 \\ L_3 \end{bmatrix},$$

and

$$g(x, y, z) = \begin{bmatrix} (y - x^2)(x^2 + y^2 + z^2 - 1)(2x - 1) \\ (xy - z)(x^2 + y^2 + z^2 - 1)(2y - 1) \\ (xz - y^2)(x^2 + y^2 + z^2 - 1)(2z - 1) \end{bmatrix}.$$

The first stage of the cascade algorithm solves the polynomial system

$$f = g + BL. \tag{7}$$

Bertini's theorem [24] provides that  $f$  has only nonsingular isolated solutions on  $\mathbb{C}[x, y, z]$ . Using a total degree homotopy, 36 paths lead to the nonsingular isolated solutions and 89 diverge to infinity. Table 1 lists the winding numbers occurring when solving  $f$  with a total degree homotopy and the average time per path for each of the winding numbers which occur. Table 2 compares the serial version of the track-back Cauchy endgame with the classical Cauchy endgame and the power series endgame. In particular, using the track-back Cauchy endgame, the endgame had to be run on only 26 of the 89 paths that diverge, resulting in less total computation time.

winding number	number of paths	Endgame	
		Cauchy	Power Series
1	41	0.010	0.007
2	4	0.092	0.084
4	72	0.023	0.156
8	8	0.734	2.641

Table 1: Distribution of the winding number for solving Eq. 7 and average time, in seconds, for running the endgame

total paths	track-back paths removed	Endgame		
		track-back Cauchy	Cauchy	Power Series
125	63	1.63	8.31	33.89

Table 2: Time, in seconds, for solving Eq. 7 using various endgames

## 4.2 A family of examples

Due to the nature of the track-back Cauchy endgame, it is more advantageous over the traditional endgame when there are many paths with large winding numbers. To illustrate this, consider a family of examples generated from [6, 25]. For  $n \geq 3$ , define

$$f_i(x_1, \dots, x_n) = x_i^n - \prod_{j \neq i} x_j \quad (8)$$

for  $i = 1, \dots, n$ . It can be shown that there are  $(n + 1)^{n-1}$  nonsingular solutions and that the origin has multiplicity  $n^n - (n + 1)^{n-1}$ , which decomposes into various cycles. Table 3 lists the winding numbers occurring for the paths that lead to the origin for  $n = 5$  and the average time per path for each of the winding numbers which occur.

Table 4 compares the other endgames with the serial version of the track-back Cauchy endgame. As  $n$  increases, there is a clear increase in the percentage of the paths that are discarded by the track-back method.

Table 5 compares the parallel version of the track-back Cauchy endgame with the classical Cauchy endgame and power series endgame. As discussed in §3.3, fewer paths were removed using the parallel version of the track-back endgame than with the serial version.

winding number	number of paths	Endgame	
		Cauchy	Power Series
4	4	0.015	0.004
5	1250	0.020	0.037
10	500	0.038	0.512
15	75	0.055	0.698

Table 3: Distribution of the winding number for solving Eq. 8 with  $n = 5$  for the paths leading to the origin and average time, in seconds, for running the endgame

n	total paths	track-back paths removed	Endgame		
			track-back Cauchy	Cauchy	Power Series
3	27	7	0.11	0.13	0.06
4	256	102	1.46	2.77	5.70
5	3125	1523	32.66	57.86	373.29
6	46,656	25,792	760.08	1662.85	10,536.13

Table 4: Time, in seconds, for solving Eq. 8 using various endgames

n	total paths	track-back paths removed	Endgame		
			track-back Cauchy	Cauchy	Power Series
5	3125	870	4.87	5.17	9.75
6	46,656	24,745	30.58	40.19	166.10

Table 5: Time, in seconds, for solving Eq. 8 in parallel using various endgames

## 5 Overview of ODE methods

Runge-Kutta methods are well-suited for use as the prediction method for homotopy continuation path tracking. They only require an initial value to start the method, do not require the evaluation of higher-order derivatives, but can still have local truncation errors of any order. Runge-Kutta methods are classified based on both the number of function evaluations and the order of the local truncation error. For example, Euler's method is the first order Runge-Kutta method requiring one function evaluation, while the classical fourth order Runge-Kutta method (RK4) requires four function evaluations.

One way to monitor the local truncation error is to evaluate the method using a stepsize  $s$  and then evaluate the method twice each with stepsize  $\frac{s}{2}$ . An error estimate is obtained by comparing the two approximations.

Another way to monitor the local truncation error is to use embedded Runge-Kutta methods. Embedded Runge-Kutta methods, developed by Fehlberg [8, 9, 10], evaluate multiple Runge-Kutta approximations simultaneously. The fifth-order Runge-Kutta-Fehlberg method (RK45) utilizes six function evaluations to compute both a fourth- and a fifth-order approximation. The difference between the two approximations provides an estimate for the local truncation error. RK45 is compared with several other embedded Runge-Kutta methods in § 7.

See [12, 20] for more information.

## 6 Adaptive stepsize and precision with higher-order methods

The adaptive precision tracking methods of [3, 4] describe rules for changing precision based on the local behavior of the path being followed. The rules provided in [3] for changing precision are based on a careful analysis of the Newton correction scheme. This analysis is extended to the Euler prediction step in [4]. Heuristics for changing the stepsize and precision together to approximate the optimal settings are also provided in that article. The following summarizes the methods of [3, 4] and extends this analysis to predictor methods with error estimates.

### 6.1 Summary of the adaptive precision methods

The adaptive precision methods of [3, 4] require the enforcement of three rules to maintain accuracy. Following the notation of those articles, let  $P$

denote the number of digits of precision and  $u = 10^{-P}$  be the unit roundoff error. Let  $10^{-\tau}$  be the accuracy to track the path and  $N$  be the maximum number of Newton iterations per step. Let  $\|\cdot\|$  be a vector norm and its induced submultiplicative matrix norm and  $\|d\|$  be the most recent error approximation (e.g., local truncation error approximation or Newton residual).

For a continuously differentiable function  $F(z) : \mathbb{C}^n \rightarrow \mathbb{C}^n$ , let  $J(z)$  denote its Jacobian matrix, i.e., the matrix of partial derivatives. Let  $\psi(z, u)$  and  $\phi(z, u)$  account for the errors in evaluating  $F(z)$  and  $J(z)$ , respectively, and suppose that they are of the form  $\psi = \Psi u$  and  $\phi = \Phi u$ . The values  $\Psi$  and  $\Phi$  can be approximated using methods presented in [3]. Let  $\mathcal{E}$  account for the growth in errors for solving a system of linear equations. Extra digits, called safety digits and denoted  $\sigma_1$  and  $\sigma_2$ , are used to account for underestimations of the values required.

The first rule asserts that the error perturbed Jacobian matrix  $J$  must be nonsingular, namely

$$P > \sigma_1 + \log_{10}[\|J^{-1}\|\mathcal{E}(\|J\| + \Phi)]. \quad (\mathbf{A})$$

With an error approximation  $\|d\|$ , the second rule asserts that the corrector must be able to converge using  $(N - i)$  Newton iterations. By letting  $D = \log_{10} [\|J^{-1}\|((2 + \mathcal{E})\|J\| + \mathcal{E}\Phi) + 1]$ , the second rule is

$$P > \sigma_1 + D + (\tau + \log_{10} \|d\|)/(N - i). \quad (\mathbf{B})$$

The final rule asserts that the final accuracy of the corrector must be smaller than the required tolerance, namely

$$P > \sigma_2 + \tau + \log_{10}(\|J^{-1}\|\Psi + \|z\|). \quad (\mathbf{C})$$

In [4], Eq. **B** is used to relate stepsize and precision. To do this, the Euler prediction is written as the initial Newton iteration creating a residual from this initial iteration that is proportional to the stepsize  $s$ , i.e.,  $\|d\| = a|s|$ . By writing  $|s| = 10^{-\xi}$ , Eq. **B** reduces to

$$P + \xi/N > \sigma_1 + D + (\tau + \log_{10} a)/N, \quad (12)$$

which can be satisfied by either increasing the precision  $P$  or decreasing stepsize by increasing  $\xi$ . The values of  $P$  and  $\xi$  are set to attempt to minimize the cost per unit advance along the path.

## 6.2 Outline of the algorithm

The adaptive precision rules presented in [3, 4] and summarized in the previous section extend to prediction methods that provide error estimates. Suppose that the prediction method computes a local error estimate of order  $p$ . That is, using a stepsize  $s$ , the local error estimate  $\|d\|$  is approximately proportional to  $|s|^{p+1}$ .

As discussed in [4], Eq. **A** is superseded by Eq. **B** upon knowing a local error estimate. Hence, Eq. **B** is applied for  $i = 0$  using the local error estimate  $\|d\|$  provided by the prediction method. If this error estimate  $\|d\|$  is smaller than the required tolerance, we accept the prediction without any Newton correction steps. Otherwise, Newton iterations are used and Eq. **B** is applied for  $i > 0$  using the Newton residual of the last iteration as the local error estimate. To validate the accuracy of the prediction, Newton iterations are also applied if there are  $M$  consecutive steps for which the prediction error is smaller than the required tolerance. In Bertini,  $M$  has a default value of 5.

Equation **B** also is used to relate stepsize and precision. Letting  $|s| = 10^{-\xi}$  and  $\|d\| = a|s|^{p+1}$ , Eq. **B** becomes

$$P + (p + 1)\xi/N > \sigma_1 + D + (\tau + \log_{10} a)/N \quad (13)$$

for the prediction (i.e.,  $i = 0$ ). The values of  $P$  and  $\xi$  are set to attempt to minimize the cost per unit advance along the path.

## 7 Computational evidence for using higher-order methods

Adaptive precision tracking using higher-order predictor methods is implemented in the software package Bertini [2]. The embedded Runge-Kutta predictor methods available in Bertini are presented in Table 6.

All the examples discussed here were run on a 2.4 GHz Opteron 250 processor with 64-bit Linux. The parallel examples were run on a cluster consisting of a manager that uses one core of a Xeon 5410 processor and 8 computing nodes each containing two 2.33 GHz quad-core Xeon 5410 processors running 64-bit Linux, i.e., one manager and 64 workers.

### 7.1 Comparing the methods

Section 5.5 of [3] and Section 3.2 of [4] describe solving a polynomial system arising from the inverse kinematics problem for a general size-revolute

Name	local error order	approximation order	# of evaluations
Heun-Euler (HE12) [12, § 8.3]	1	2	2
Norsett (RKN34) [7]	3	4	5
Fehlberg (RKF45) [12, § 8.3]	4	5	6
Cash-Karp (RKCK45) [5]	4	5	6
Dormand-Prince (RKDP56) [19]	5	6	8
Verner (RKV67) [26]	6	7	10

Table 6: Summary of embedded Runge-Kutta methods implemented in Bertini

ODE method	96 bit fixed precision	method of [3]	method of [4]	
			time	nfe/path
Euler	184.01	38.54	32.73	1296

  

ODE method	96 bit fixed precision	new method	nfe/path
HE12	80.12	17.97	550
RKN34	62.21	16.96	454
RKF45	55.01	14.48	445
RKCK45	49.13	14.13	403
RKDP56	56.50	16.41	461
RKV67	66.46	16.64	620

Table 7: Comparison of the average time of 10 runs of the IPP system, in seconds

serial-link robot [15]. Using the same settings, Table 7 compares the methods of [3] and [4], using the minimum fixed precision (96 bits) with various predictor methods, and the adaptive precision method of this paper. The column labeled “nfe/path” presents the average number of function evaluations per path. For each function evaluation, there is an associated set of linear equations to solve. Together, function evaluation and solving the associated linear equations are by far the most expensive part of homotopy continuation.

## 7.2 A family of systems

An example of a family of systems where every path leads to a nonsingular solution is the family of economics problems derived from [14, § 7]. The

$n$	paths	HE12	RKN34	RKF45	RKCK45	RKDP56	RKV67
10	256	2.5s	1.7s	1.7s	1.4s	1.8s	2.1s
11	512	4.6s	3.2s	3.2s	2.6s	3.5s	4.6s
12	1024	11.8s	8.5s	8.7s	6.9s	9.2s	11.0s
13	2048	32.0s	21.4s	21.6s	16.9s	24.1s	28.9s
14	4096	1m32s	1m9s	1m12s	54.1s	1m21s	1m35s
15	8192	2m49s	2m4s	2m10s	1m39s	2m22s	2m46s
16	16,384	7m3s	5m7s	5m25s	4m15s	6m2s	7m12s
17	32,768	16m46s	12m50s	14m11s	10m58s	15m49s	19m33s
18	65,536	42m3s	30m52s	34m19s	25m21s	36m48s	52m6s

Table 8: Comparison for solving  $F_n$  using serial processing

original presentation of the polynomial system is

$$G_n(x_1, \dots, x_n) = \begin{bmatrix} x_{n-1}x_n - 1 \\ (x_{n-2} + x_1x_{n-1})x_n - 1 \\ (x_{n-3} + x_1x_{n-2} + x_2x_{n-1})x_n - 1 \\ \vdots \\ (x_1 + x_1x_2 + \dots + x_{n-2}x_{n-1})x_n - 1 \\ x_1 + x_2 + \dots + x_{n-1} + 1 \end{bmatrix} = 0. \quad (14)$$

This system is reduced in [14] by noting that the first equation implies that  $x_{n-1} \neq 0$  so that  $x_n = \frac{1}{x_{n-1}}$ . Upon substitution and clearing denominators for the remaining equations, we obtain the polynomial system

$$F_n(x_1, \dots, x_{n-1}) = \begin{bmatrix} x_{n-2} + x_1x_{n-1} - x_{n-1} \\ x_{n-3} + x_1x_{n-2} + x_2x_{n-1} - x_{n-1} \\ \vdots \\ x_1 + x_1x_2 + \dots + x_{n-2}x_{n-1} - x_{n-1} \\ x_1 + x_2 + \dots + x_{n-1} + 1 \end{bmatrix} = 0. \quad (15)$$

For all  $n$ ,  $F_n$  has total degree  $2^{n-2}$ , which is equal to the number of nonsingular solutions.

The system  $F_n$  for  $n = 10, \dots, 18$  was solved using various prediction methods with a tracking tolerance of  $10^{-7}$  and a final endgame convergence tolerance of  $10^{-10}$ . The results are summarized in Table 8.

ODE method	time
HE12	8.48
RKN34	5.93
RKF45	6.32
RKCK45	5.73
RKDP56	7.02
RKV67	8.77

Table 9: Comparison for solving nine-point path synthesis problem, in minutes

### 7.3 Solving a large system

Section 3.3 of [4] describes solving a polynomial system arising from the nine-point path synthesis problem [27] using a homotopy that utilizes its 2-homogeneous structure and two-fold symmetry. This system was solved using various prediction methods with a tracking tolerance of  $10^{-7}$  and a final endgame convergence tolerance of  $10^{-10}$  in parallel. The results are summarized in Table 9.

## 8 Conclusions

This article introduces two ways in which predictor/corrector methods may be made more efficient, at least in the algebraic setting. The first method – the track-back Cauchy endgame – reduces the number of times the Cauchy endgame is run. Homotopies which have endpoints with large winding numbers will benefit most from this advance, as seen in the examples above.

The second method – the use of Runge-Kutta methods in homotopy continuation – is not new in its own right, but the addition of adaptive precision techniques to Runge-Kutta methods is new. This article shows that higher order methods *are* worth using in this setting, debunking folklore in the numerical algebraic geometry community.

This article will by no means be the final article on efficiency in homotopy continuation; much remains to be studied. For example, multistep methods are not covered in this article but would likely show further savings. Path tracking can also be made more efficient via parallelization. Only the most basic parallel operations have been studied thus far, so this is another avenue of future research related to this article. Finally, some of the ideas of this article, particularly regarding higher-order methods, may carry over to the case of homotopy continuation for general nonlinear functions.

## References

- [1] E.L. Allgower and K. Georg. Numerical continuation methods, An introduction. Springer Series in Computational Mathematics, vol. 13, Springer-Verlag, Berlin, 1990.
- [2] D.J. Bates, J.D. Hauenstein, A.J. Sommese, and C.W. Wampler. Bertini: Software for Numerical Algebraic Geometry. Available at <http://www.nd.edu/~sommese/bertini>.
- [3] D.J. Bates, J.D. Hauenstein, A.J. Sommese, and C.W. Wampler, Adaptive multiprecision path tracking, *SIAM J. Numer. Anal.*, 46:722–746, 2008.
- [4] D.J. Bates, J.D. Hauenstein, A.J. Sommese, and C.W. Wampler. Step-size control for adaptive multiprecision path tracking. To appear in *Interactions of Classical and Numerical Algebraic Geometry*, D. Bates, G. Besana, S. Di Rocco, and C. Wampler (eds.), *Contemporary Mathematics*, 2009.
- [5] J.R. Cash and A.H. Karp. A variable order Runge-Kutta method for initial value problems with rapidly varying right-hand sides. *ACM Trans. Math. Software*, 16(3): 201–222, 1990.
- [6] B. Dayton and Z. Zeng. Computing the multiplicity structure in solving polynomial systems. Proceedings of ISSAC 2005 (Beijing, China), 116–123, 2005.
- [7] W.H. Enright, K.R. Jackson, S.P. Nørsett, and P.G. Thomsen. Interpolants for Runge-Kutta formulas. *ACM Trans. Math. Software*, 12(3):193–218, 1986.
- [8] E. Fehlberg. Klassische Runge-Kutta-Formeln fünfter und siebenter Ordnung mit Schrittweiten-Kontrolle. *Computing (Arch. Elektron. Rechnen)*, 4:93–106, 1969.
- [9] E. Fehlberg. Klassische Runge-Kutta-Formeln vierter und niedrigerer Ordnung mit Schrittweiten-Kontrolle und ihre Anwendung auf Wärmeleitungsprobleme. *Computing (Arch. Elektron. Rechnen)*, 6:65–71, 1970.
- [10] E. Fehlberg. Low-order classical Runge-Kutta formulas with stepsize control and their application to some heat transfer problems. NASA Technical Report 315.

- [11] B. Huber and J. Verschelde. Polyhedral end games for polynomial continuation. *Numer. Algorithms*, 18(1): 91–108, 1998.
- [12] D. Kincaid and W. Cheney. *Numerical analysis : mathematics of scientific computing*. Brooks/Cole Publishing Co., Pacific Grove, CA, third edition, 2002.
- [13] T.Y. Li. Numerical solution of polynomial systems by homotopy continuation methods. In *Handbook of Numerical Analysis, Volume XI, Special Volume: Foundations of Computational Mathematics*, F. Cucker, ed., North-Holland, 2003, 209–304.
- [14] A. Morgan. *Solving polynomial systems using continuation for engineering and scientific problems*. Prentice Hall Inc., Englewood Cliffs, NJ, 1987.
- [15] A.P. Morgan and A.J. Sommese. Computing all solutions to polynomial systems using homotopy continuation. *Appl. Math. Comput.*, 24(2): 115–138, 1987. Errata: *Appl. Math. Comput.*, 51:209, 1992.
- [16] A.P. Morgan, A.J. Sommese, and C.W. Wampler. Computing singular solutions to nonlinear analytic systems. *Numerische Math.*, 58:669–684, 1991.
- [17] A.P. Morgan, A.J. Sommese, and C.W. Wampler. A power series method for computing singular solutions to nonlinear analytic systems. *Numerische Math.*, 63:391–409, 1992.
- [18] A.P. Morgan, A.J. Sommese, and C.W. Wampler. Computing singular solutions to polynomial systems. *Adv. in Appl. Math.*, 13:305–327, 1992.
- [19] P.J. Prince and J.R. Dormand. High order embedded Runge-Kutta formulae. *J. Comput. Appl. Math.*, 7(1):67–75, 1981.
- [20] L.F. Shampine. *Numerical solution of ordinary differential equations*. Chapman & Hall, New York, 1994.
- [21] A.J. Sommese and J. Verschelde. Numerical homotopies to compute generic points on positive dimensional algebraic sets. *J. Complexity*, 16(3):572–602, 2000.
- [22] A.J. Sommese, J. Verschelde, and C.W. Wampler. Using monodromy to decompose solution sets of polynomial systems into irreducible components. *Proceedings of the 2001 NATO Advance Research Conference (Eilat, Israel)*, 297–315, 2001.

- [23] 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.
- [24] A. Sommese and C. Wampler. *The Numerical Solution to Systems of Polynomials Arising in Engineering and Science*. World Scientific, Singapore, 2005.
- [25] B. Sturmfels. *Solving Systems of Polynomial Equations*. Number 97 in CBMS Regional Conference Series in Mathematics, AMS, 2002.
- [26] J.H. Verner. Explicit Runge-Kutta methods with estimates of the local truncation error, *SIAM J. Numer. Anal.*, 15(4):772–790, 1978.
- [27] C.W. Wampler, A. Morgan, and A.J. Sommese. Complete solution of the nine-point path synthesis problem for four-bar linkages, *ASME Journal of Mechanical Design* 114(1): 153–159, 1992.