Bertini_real: Numerical Decomposition of Real Algebraic Curves and Surfaces

Daniel A. Brake*,[1], Daniel J. Bates[2], Wenrui Hao[3],
Jonathan D. Hauenstein[1], Andrew J. Sommese[1], Charles W. Wampler[4]
[1] Department of Applied and Computational Mathematics and Statistics, University of Notre Dame
[2] Department of Mathematics, Colorado State University
[3] Mathematical Biosciences Institute
[4] General Motors R&D
*Corresponding author; email: danielthebrake@gmail.com

February 17, 2015

Abstract

Bertini_real is a compiled command line program for numerically decomposing the real portion of a positive-dimensional complex component of an algebraic set. The software uses homotopy continuation to solve a series of systems via regeneration from a witness set to compute a cell decomposition. This decomposition captures the topological information and permits further computation on the real sets, such as sampling, visualization, and 3D printing.

Keywords: Numerical algebraic geometry, homotopy continuation, polynomial system, real solutions, cell decompositions.

1 Introduction

The solution sets of systems of *linear* equations are easily described by providing the appropriate number of basis vectors. Visualization of such linear spaces is of little interest as two sets of the same dimension look the same, up to rotation and transposition. On the contrary, the solution sets of systems of nonlinear *polynomial* equations are much more difficult to describe, and visualization of such sets has value in numerous settings. This article presents an implementation, Bertini_real, of a novel numerical method for finding and visualizing real curves and surfaces that are defined by systems of polynomial equations, with no theoretical upper bound on the dimension of the ambient real Euclidean space. As motivation, the Bertini_real output for one such surface in $\mathbb{R}^3$ is given in Figure 1.

While the methods of this article are certainly not the first approach to visualizing real algebraic sets, they produce data and images in a novel way, with various advantages and disadvantages in comparison to existing methods. Three of the best known methods for working with real algebraic sets are the *cylindrical algebraic decomposition*, *discretization into meshes*, and *ray tracing*. We next describe each of these other techniques briefly, indicating the similarities and differences of our methods.

The cylindrical algebraic decomposition (CAD) of a real algebraic set is a decomposition of the ambient Euclidean space into connected semialgebraic sets (regions of space demarcated by algebraic sets) so that each polynomial has a constant sign on each semialgebraic set. For example, a sphere

This work was partially supported by the Duncan Chair of the University of Notre Dame, AFOSR grant FA8650-13-1-7317, NSF DMS-1115668, NSF DMS-1262428, DARPA YFA, Sloan Research Fellowship, and the Mathematical Biosciences Institute.

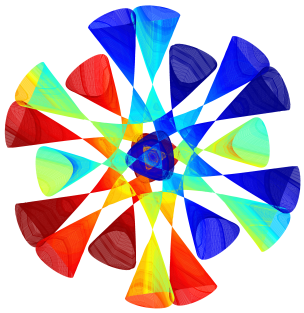

Figure 1: Decomposition of the Barth Sextic, a real algebraic surface, computed with Bertini_real.

in $\mathbb{R}^3$ is decomposed into the region outside the sphere, that within the sphere, and a number of cells of varying dimensions on the sphere itself. Techniques for computing such a decomposition go back to [18] and are described thoroughly in [5]. See also [4]. CAD methods are often algebraic or symbolic in nature, yielding algebraic or symbolic output, which is inherently different from our highly geometric numerical cellular decomposition. For example, a one-dimensional CAD cell is described by a set of equalities and inequalities with little apparent indication of geometric structure (aside from dimension). A cell in our decomposition consists of boundary points and a general point in the middle, which can be moved (using numerical continuation) to trace out the shape.

Meshes are used throughout mathematics and applications, prominently in finite element methods [17]. They also are used in the computational geometry library CGAL [34], in theoretical developments in sources such as [1], and in smoothing of preexisting meshes as in [19]. The idea here is to discretize a surface into a number of polygons and use this decomposition for further computations or visualization. The use of discretization necessarily turns a fundamentally continuous, geometric object into a discrete, combinatorial object.

Finally, ray tracing, used frequently in computer graphics, is a means of producing a two-dimensional image of a three-dimensional object by computing which rays originating from an imagined camera intersect with the object [21]. The history of this geometric concept likely goes back at least to [3], though may trace its roots back centuries in some form. The key point for this article is that ray tracing yields knowledge of only some portion of the object being observed, that which is "visible" to the camera. For some forms of graphics and imaging, this is surely adequate, though ray tracing certainly does not yield full knowledge of the object being observed. As a result, the decomposition of this article yields more complete information than ray tracing, and can handle more geometric difficulties (self-crossings, singularity of whole components, etc.) than previously developed techniques.

Bertini_real takes as input a set of polynomial equations (much like the CAD) and uses recently developed numerical geometric methods rooted in algebraic geometry to produce output that can be used for visualization or further computations on algebraic curves and surfaces in (theoretically) any ambient real Euclidean space. More specifically, the algorithms of *numerical algebraic geometry* are employed, namely homotopy continuation. These probability-one algorithms for solving polynomial systems and manipulating their solution sets have matured over the past few decades, resulting in trustworthy, well-tested techniques and software, particularly Bertini [8]. These methods are described in adequate detail in the next section.

The output of our program is Morse-theoretic in nature, described here in a simple way. See §2.6 for a more complete definition of our data types. For computed points, we provide a numerical

approximation to the point. As with any points produced with numerical algebraic geometry methods, we can find as many digits of accuracy as the user might desire. For curves, we decompose into edges by making use of a single real linear projection, with each edge represented by numerical approximations of endpoints (either a critical point or a smooth point in the same projection fiber as a critical point) and a numerical approximation of a point somewhere between. Surfaces are decomposed using two linear projections and are represented by faces. Each face consists of a number of boundary edges and a numerical approximation of a point within the face. This decomposition is not new and has been used, for example, in CGAL [29].

In §2 we lay out the preliminaries necessary to understand the body of the paper, including basics of homotopy continuation and the numerical irreducible decomposition. Because the surface decomposition method relies on repeated curve decomposition, the curve method appears first in §3, with surfaces coming next in §4. Some Bertini_real-specific implementation choices are described in §5. Demonstrative examples are provided in §6.

It should be noted that this is the latest in a string of articles on the use of numerical algebraic geometry to decompose real algebraic sets. The curve case was described in [28] and a less complete surface method was given in [13]. More recently, an extended abstract [15] describes some of this material with minimal details, and another describes the removal of the almost smooth condition [7].

2 Preliminaries

In this section, we briefly discuss the necessary items to describe the implementation of Bertini_real. These are discussed in much greater detail in [13], and in the sources indicated below.

2.1 Homotopy Continuation

The fundamental numerical method for virtually every computation in this paper is *homotopy continuation*. The goal is to solve the system of polynomial equations $f = 0$. This is accomplished by solving a related system of equations $g = 0$ that has some of the structure of f. The system g is then deformed continuously (in fact, differentiably) into f and the solutions of $g = 0$ are tracked through this deformation using predictor-corrector methods. With proper construction, we are theoretically guaranteed to obtain a superset of all zero-dimensional solutions to $f = 0$ (points), which can easily [10] be trimmed down to the set of zero-dimensional solutions. Figure 2 provides a schematic view of homotopy continuation.

There are many ways to form g, the most basic of which is the *total degree* start system. Other systems include m-homogeneous, linear product, and polyhedral methods. See [33, Chap. 8] for a general overview. General references for homotopy continuation include [12, 33].

2.2 Witness Sets and Numerical Irreducible Decomposition

Algebraic sets – solution sets of polynomial systems – may have complicated structure in that they may have many *irreducible components* of varying dimension, degree, and multiplicity structure. One can geometrically decompose each algebraic set into its irreducible components, thereby producing a numerical description of each component via a *witness set*. Such a numerical description of the set of all irreducible components of an algebraic set is called a *numerical irreducible decomposition* [30].

For an irreducible component $C \subset \mathbb{C}^N$, a witness set for C incorporates information about the dimension and degree of C as well as a polynomial system f such that C is an irreducible

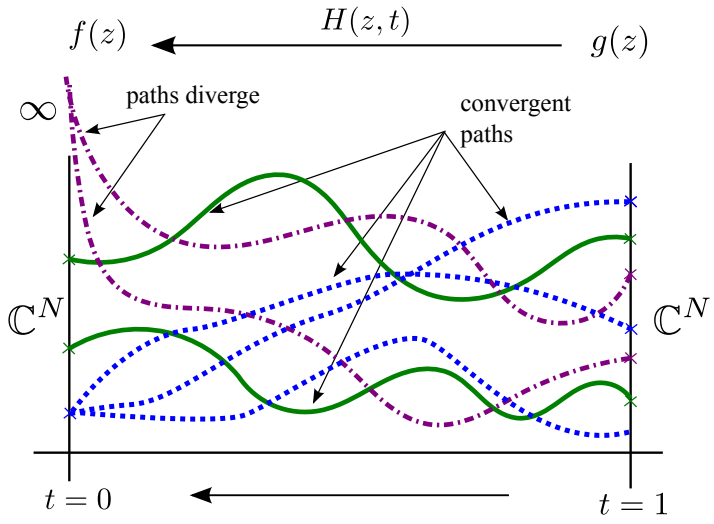

Figure 2: Generic homotopy continuation. Move $H(z,t)$ from $t=1$ to $t=0$, deforming $g(z)$ to $f(z)$, following a predictor-corrector path, employing endgame methods to approach $t=0$, allowing computation of singular endpoints. Computations take place over a projectivization of $\mathbb{C}^N$, being the algebraic closure of $\mathbb{R}^N$.

component of

$$\mathcal{V}(f) = \{x \in \mathbb{C}^N \mid f(x) = 0\}.$$

In short, a witness set for the irreducible component C is a triple $\mathcal{W} = \{f, \mathcal{L}, W\}$ where $\mathcal{L} \subset \mathbb{C}^N$ is a general linear space of codimension equal to the dimension of C and $W = C \cap \mathcal{L}$ with deg $C = |W|$. The system f, linear space $\mathcal{L}$, and set of points W are called a *witness system*, *witness slice*, and *witness point set* for C, respectively.

2.3 Deflation

The tracking of solution paths in homotopy continuation is performed using predictor-corrector approaches, e.g., see [11]. This requires the Jacobian to be invertible along the path, except possibly at the endpoint. Since we will be tracking along components using witness sets, we will require that the component have multiplicity 1 with respect to its witness system. When the component has multiplicity > 1 with respect to its witness system, we will simply replace the witness system by a new polynomial system constructed using isosingular deflation [24] to ensure multiplicity 1 and nonsingularity to allow for predictor-corrector tracking on the component.

2.4 Randomization

In order to maintain numerical stability when performing deformations, we will always deform square systems, that is, systems with the same number of polynomial equations and variables. To ensure this, Bertini_real employs randomization. In particular, we will enforce that all witness systems have the same number of polynomials as the codimension of the irreducible component. If a witness system f has more polynomials than the codimension, we will replace f by $R \cdot f$ where

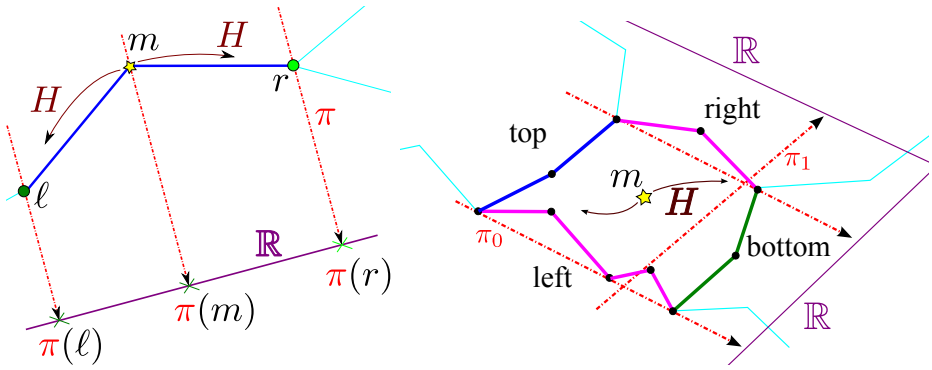

Figure 3: The edge (left) and face (right) data types for the numerical cellular decomposition. The edge is equipped with three points, ℓ, m, and r. Coupled with projection $\pi(x)$ and the system which defines the curve, the edge has a homotopy H which may be used to move the generic midpoint m around, so long as the projection value does not cross $\pi(\ell)$ or $\pi(r)$. The edge probably is connected to other edges, colored in cyan here. A face is bounded by top, bottom, left, and right edges, and is itself equipped with a trackable midpoint and homotopy. Faces are connected to other faces via sharing of edges.

R is a random matrix with the number of rows equal to the codimension. Bertini's Theorem [33, Theorem A.8.7] ensures that $R \cdot f$ is also a witness system.

2.5 Regeneration

Regeneration [22] is an incremental technique for solving a system of polynomial equations $f = 0$ one equation at a time. Below, we will apply a linear product regeneration to solve various systems starting with a witness set for a component of interest. In particular, a witness point set and witness slice will be used as the starting point of regeneration to compute points of interest, e.g., critical points. Bertini_real uses various custom implementations of regeneration to perform the necessary computations.

2.6 Edges and Faces

Curve decomposition requires the use of one generic linear projection π, while surface decomposition requires the use of two, π_0 and π_1. The two data types Bertini_real creates when decomposing curves and surfaces are *edges* and *faces*, as 1- and 2-cells, appearing in Figure 3. The point is the 0-cell, which needs no further explanation.

The *edge* is the fundamental one-dimensional cell building block for our decompositions. Each edge has three points, ℓ, m, and r, as the left, mid, and right points. The two endpoints ℓ and r lie in the π fiber of some critical point, which may be the endpoint itself. The midpoint m is generic in the sense that it cannot be a critical point, and therefore is trackable in a homotopy (appearing below as (6) in §3.1.4) which changes the value of $\pi(m)$ to be anywhere in the interval $(\pi(\ell), \pi(r))$. An edge is *degenerate* if $\ell = m = r$. See the left of Figure 3.

A *face* builds on an edge, to again have boundaries and a trackable midpoint. The boundaries of a 2-cell are edges lying on curves, possibly degenerate edges. There are four boundaries for a face: top, bottom, left, and right. The left and right edges have constant π_0 projection value, and meet with the top and bottom edges at endpoints. There may be more than one edge per left and

right boundary. On top and bottom, edges do not have constant projection value, but separate regions which would otherwise perhaps repeat π_0 values. That is, top and bottom edges tend to lie on critical curves. The face is equipped with a midpoint which can be moved around using a homotopy H, so long as it is not tracked across π_0 for the left and right edges, and π_1 for the top and bottom. The homotopy for this (10) appears below in §4.1.4. See the right side of Figure 3.

3 Curve Decomposition

The ability to describe an algebraic curve is important in its own right, as well as being a necessary ingredient for the surface decomposition appearing in §4. Bertini_real implements the algorithm from [28] in such a fashion that a user may decompose the real points of a complex algebraic curve in any dimensions as well as decomposing many curves in the course of decomposing a surface.

Let $C \subset \mathbb{C}^N$ be a complex algebraic curve, that is, a one-dimensional complex component of $\mathcal{V}(f)$ for some polynomial system f, where $\mathcal{V}(f)$ is the complex solution set of $f = 0$. The real points of the curve C, namely $C \cap \mathbb{R}^N$, will be decomposed with respect to a sufficiently general real linear projection $\pi_0 : \mathbb{R}^N \to \mathbb{R}$. That is, $C \cap \mathbb{R}^N$ will be decomposed into cells so that each cell can be parameterized by π_0 over a line segment in $\mathbb{R}$. These segments are separated by projections of critical points under the projection π_0.

3.1 Six Steps to Curve Decomposition

The curve decomposition consists of six steps that are described later in this section:

1. Find critical points.
2. Intersect with sphere.
3. Slice at and between critical points.
4. Connect midpoints to critical points.
5. Merge away superfluous edges.
6. Refine the decomposition.

3.1.1 Solving for Critical Points

The fundamental step of the curve cellular decomposition method is to compute the *critical points* of $C \cap \mathbb{R}^N$ with respect to the projection π_0. The critical points will be used to identify where the parameterization has to be changed. That is, away from the critical points, the curve is smooth and parameterized by π_0. The critical points of C are those points on $C \cap \mathbb{R}^N$ for which the Jacobian matrix of the witness system f for C, concatenated with the direction of π_0, is rank-deficient. These are precisely the points in $C \cap \mathbb{R}^N$ which are solutions to

$$\begin{bmatrix} f(x) \\ \det \begin{pmatrix} Jf(x) \\ J\pi_0 \end{pmatrix} \end{bmatrix} = 0. \tag{1}$$

To obtain the critical points, we perform regeneration starting from a witness set for C computed by `Bertini` [8]. A summary diagram for this computation is provided in Figure 4. Since the determinant in (1) is potentially of high degree, Bertini_real computes the critical points using a

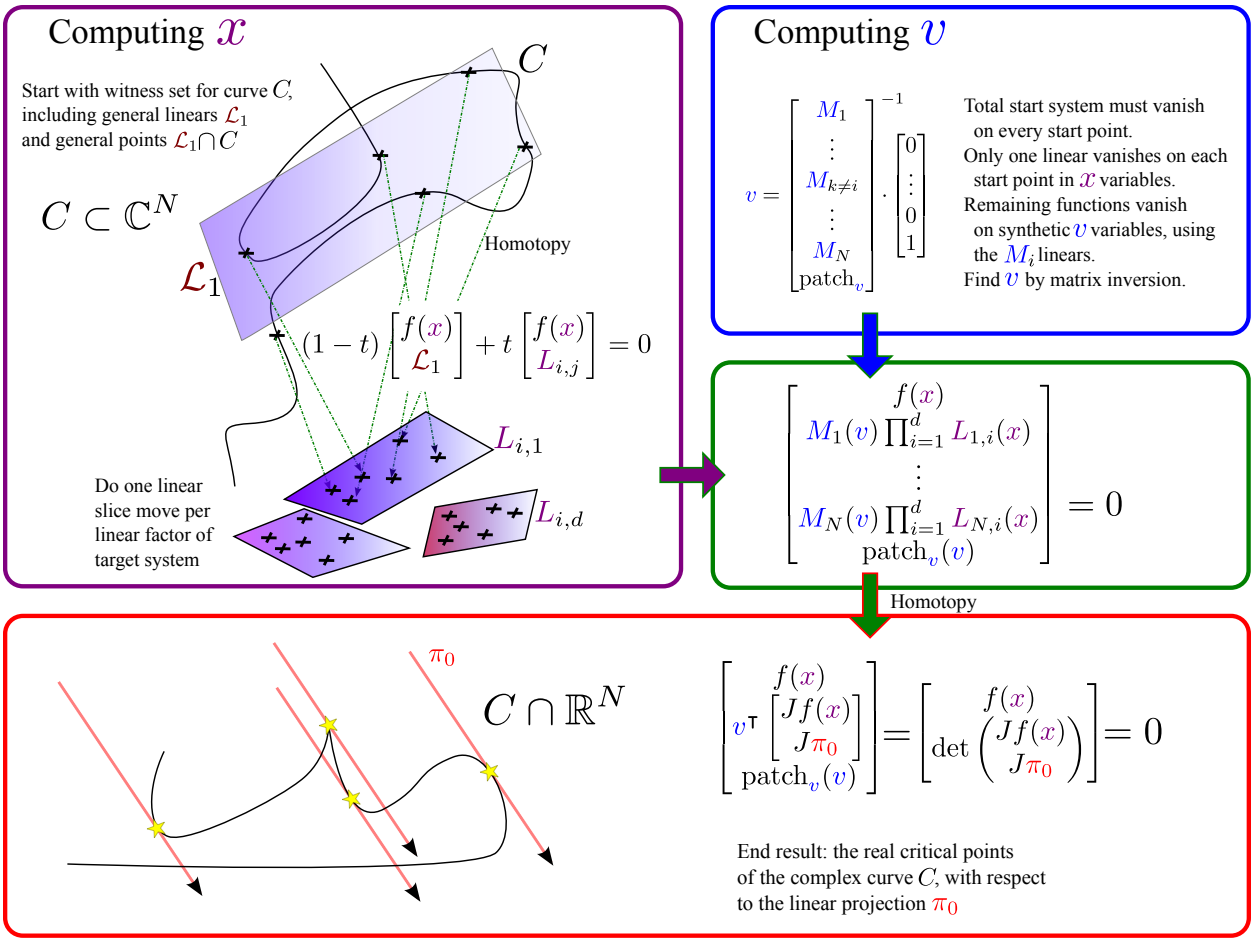

Figure 4: Solving for critical points of a curve C, indicated by yellow stars. We build up to a 2-homogeneous start system using two subroutines. First, we track through a homotopy from $\mathcal{L}_1$ to other random complex linear equations, only in the natural x variables. Then, we invert matrices consisting of stacked linear equations, to obtain the values of the synthetic v variables. Finally, we move the concatenation $[x, v]$ through a homotopy to find the critical points of C.

null space formulation [9]. That is, Bertini_real introduces new variables v which will be an element in the left null space of the matrix in (1). The resulting homotopy for computing the critical points is:

$$(1-t)\begin{bmatrix} f(x) \\ v^\intercal \begin{bmatrix} Jf(x) \\ J\pi_0 \end{bmatrix} \\ \text{patch}_v(v) \end{bmatrix} + t\begin{bmatrix} f(x) \\ M_1(v)\prod_{i=1}^{d-1} L_{1,i}(x) \\ \vdots \\ M_N(v)\prod_{i=1}^{d-1} L_{N,i}(x) \\ \text{patch}_v(v) \end{bmatrix} = 0 \tag{2}$$

where the start points of this homotopy are computed via regeneration described below. Here, d is the maximum degree of any polynomial in f. Since C has multiplicity 1 with respect to f and f has been randomized to $N-1$ polynomials in N variables, the homotopy described in (2) is square and there are at most finitely many critical points on C, namely $x \in C$ which solve (1). It

is possible that the fiber of the projection from $(x, v) \to x$ is positive dimensional at some of the critical points, but [6] shows that, for each critical point x, there exists v such that (x, v) is an endpoint of the homotopy (2).

The start points to (2) are obtained using two types of solves – one in x and one in v. To obtain the x coordinates, we compute the witness point set of C with respect to each of the hypersurfaces $\mathcal{V}(L_{i,j})$ one at a time by using the original witness slice and witness point set. Then, since the start system must vanish at the point we are constructing, and only one $L_{i,j}$ can vanish per start point, each M_k with $k \neq i$ must vanish. Hence, the second solve involves simply using numerical linear algebra by forming a matrix from the appropriate M linear equations, and the v-patch linear, and invert:

$$v_{\text{start}} = \begin{bmatrix} M_1 \\ \vdots \\ M_{k \neq i} \\ \vdots \\ M_N \\ \text{patch}_v \end{bmatrix}^{-1} \cdot \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix} \tag{3}$$

Once we have the start variables in x and v, we simply concatenate them so that our start points look like $s_{i,j} = (x_{i,j}; v_i)$, and the start system vanishes on all $s_{i,j}$. Then we perform the homotopy movement to solve the left nullspace system, to obtain the critical points. As we are performing a decomposition of the *real* portion of the component, so in post-processing we discard the complex solutions. The remainder is the set of real critical points χ, which we commit to the vertex set, which is one of the main outputs of Bertini_real.

3.1.2 Sphere Intersection

We know the interesting parts of the curve will occur at χ, but there may be parts of C which head to infinity, and we must capture this information. To do so, we intersect C with a sphere of radius r and center c.

The choice of a *bounding sphere* as opposed to a bounding box is somewhat arbitrary. We chose a sphere because it is nicely represented in a single equation of degree two. The other obvious choice for capturing divergent behavior would be a bounding box since it requires no regeneration and adapts naturally to many visualization schemes. However, a box yields two intersection computations per variable in the problem (one per each end of the bounding interval), which initially increased the complexity of the code while simultaneously reducing its readability. Hence, we opted for a sphere.

To ensure the bounding sphere contains the important parts of the curve (the parts around the critical points), we let the sphere be centered at the centroid of the critical points and let its radius be three times the maximum distance from the center to any critical point. We compute the intersection points via a simple two-step regeneration similar to (2). First we track the supplied general witness linear $\mathcal{L}$ to two other general linear equations L_1, L_2 (because the degree of the sphere function is two), and then solve this homotopy:

$$(1-t)\begin{bmatrix} f(x) \\ ||x-c||_2^2 - r^2 \end{bmatrix} + t\begin{bmatrix} f(x) \\ L_1(x) \cdot L_2(x) \end{bmatrix} = 0 \tag{4}$$

We discard any complex solutions found.

Should the user desire to limit their view of C to a specific region of the domain, rather than use an automatically computed sphere, they may supply at runtime a plaintext file containing r

and c. If the curve being decomposed is part of another decomposition, C inherits c and r from that decomposition.

3.1.3 Bisection and Slicing

Having found χ, the set of points where interesting things happen, the next step is to discover midpoints over the intervening intervals. Hence, we form the ordered set of projection values of the critical points, $\pi_0(\chi) = \{p_{c_j}\}$, checked for uniqueness up to a numerical tolerance, and the corresponding midpoints of projection value intervals, $\{p_{m_i}\} = \{(p_{c_i} + p_{c_{i+1}})/2\}$.

We must find the points lying 'above' the bisector of the interval in terms of projection value. For each p_{m_i}, move the general witness linear $\mathcal{L}$ to the linear $\pi_0(x) - p_{m_i}$ in order to obtain midpoints 'upstairs', using the homotopy

$$(1-t)\begin{bmatrix} f(x) \\ \pi_0(x) - p_{m_i} \end{bmatrix} + t\begin{bmatrix} f(x) \\ \mathcal{L} \end{bmatrix} = 0. \tag{5}$$

Since we are performing a real decomposition, we discard any complex solutions.

We retain the association between the midpoints and which interval they came from, as in the next step we seek to discover the connections to the critical points.

3.1.4 Connecting the Dots

In this step, the midpoints are tracked to critical points to form the edges of the decomposition.

For each midpoint, we need to find its neighbors to the left and right. We do this by once again performing a homotopy, this time only in terms of projection value. For $j = i, i+1$, move $\pi_0(x) - p_{m_i}$ to $\pi_0(x) - p_{c_j}$ using

$$(1-t)\begin{bmatrix} f(x) \\ \pi_0(x) - p_{c_j} \end{bmatrix} + t\begin{bmatrix} f(x) \\ \pi_0(x) - p_{m_i} \end{bmatrix} = 0 \tag{6}$$

Since all the singularities of this homotopy occur in the fiber over p_{c_j}, this homotopy is guaranteed to be singularity-free, except at the endpoints.

It is possible that a midpoint does not track to a known point – when the edge it lies on does not go critical at the corresponding projection value but some other edge does. Here, we will find heretofore undiscovered points, and we can flag them as 'new', for removal in the merge step.

3.1.5 Merge

As an optional step to the curve algorithm (though it is mandatory either to use or not use for certain curve decompositions produced in a surface decomposition), we can merge edges. The edges which are candidates for merging are those which have 'new' endpoints as found in the connection step. We use a slightly optimized merge method, illustrated in Figure 5.

First we find merge candidates. Of the edges in the decomposition, find those with 'new' right point, and iterate to the right along connecting edges, keeping an ordered list of edges to merge together, until the right edge is no longer new. Then, we track a homotopy in the style of (6) from the most middle of the midpoints to a new midpoint having projection value equal to the average projection value of two outer endpoints. This process is repeated until all candidate edges have been merged. The optimization made is in merging as many edges as possible simultaneously, rather than merging one pair at a time.

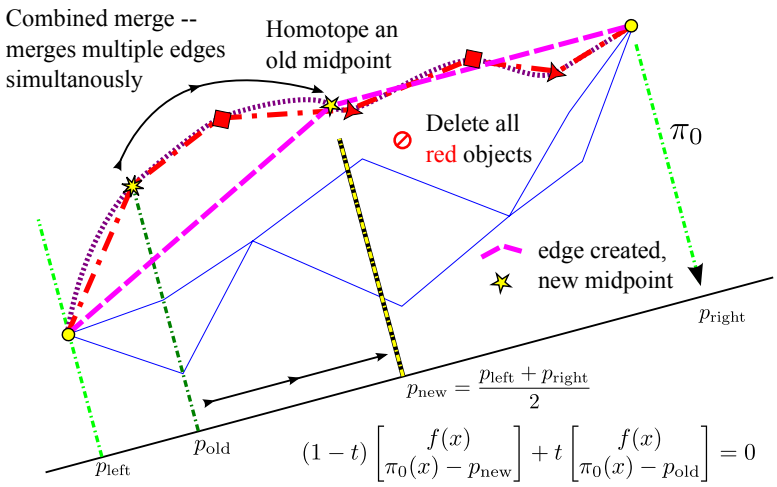

Figure 5: Merging edges in Bertini_real. 'New' endpoints (red squares) for edges can be removed, by finding the chain of edges such that the left and right (yellow circles) are 'critical', and all the interior endpoints are 'new'. Averaging the projection values of the critical endpoints produces the projection value for the new endpoint, which itself is obtained by homotoping an old midpoint to the new (yellow star).

3.1.6 Refinement

One could easily end the decomposition at this point. We have a complete characterization of C, including all critical points, topological information of how the branches of the curve meet at the critical points, and how the curve goes to infinity. However, if one wanted to perform integration of some function over the curve, for instance, or simply visualize it, they probably will want to refine the data. Fortunately, since each midpoint is equipped not merely with coordinates but also a domain of validity in terms of the π_0 values of the right and left points *and* an explicit trackable singularity-free homotopy, we can almost trivially refine the curve for further tasks.

The refinement process amounts to moving linear equations once again, where the linear to be moved is the projection π_0. This is the beauty of using linear projections to perform the decomposition – linear moves are computationally cheap.

Bertini_real provides two refinement routines in its supplementary `Sampler` program – a fixed-number method, where each edge will have a uniform number of samples spaced evenly in terms of projection value, and an adaptive method, whereby samples are added to reduce the length of adjoining line segments until their lengths are below a threshold.

Fixed Curve Sampler The simple fixed-number curve sampler is provided for two reasons. First, it is very fast to run and the user directly specifies the number of points. More importantly, however, is the fact that the current surface refinement method uses it to sample curve decompositions for certain curves on the surface.

As input, the user provides a number of samples μ they would like per edge. This number includes the two end points, so that the number in the middle of the edge is $\mu - 2$, and choosing $\mu = 3$ does not refine the curve at all since every edge already has three points on it. The default setting is $\mu = 7$.

Edges of Bertini_real curve decompositions can be of greatly varying size, especially if the curve is not compact. The fixed-number sampler therefore produces blocky results which are unsightly

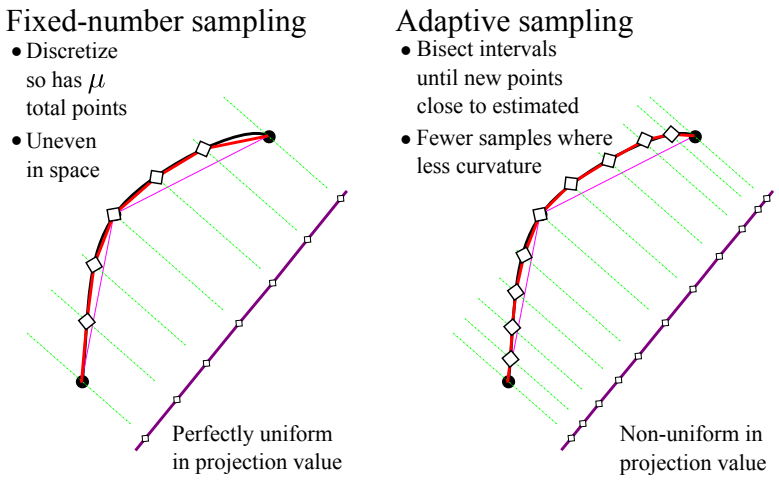

Figure 6: Bertini_real curve refinement methods, fixed-number on the left, and distance-adaptive on the right.

and unsuitable for further computations. The provided adaptive sampling method improves on these shortfalls.

Adaptive Curve Sampler The adaptive curve sampling method is iterative, and uses an estimate-bisect pattern to produce a sampling of a curve. The method allows the user to set two parameters for refinement – a distance threshold ε such that `Sampler` seeks to refine until the distance between sample and estimated point are separated by less than ε, and a maximum number of refinement iterations M.

Each refinement pass undergoes the following process. For each interval between existing samples x_i and x_{i+1} on an edge, compute the average point. This is the estimated next sample, and it has a projection value of $\pi_0(\frac{1}{2}(x_i + x_{i+1}))$. We homotope in terms of π_0 from a generic point on the edge to the new projection value, and obtain a new sample point. If the distance between the new and estimated points is less than ε, then the two resulting intervals do not need to be refined on the next iteration. Otherwise, we flag them for refinement on the next pass. This refinement loop continues until no intervals are flagged for refinement, or M (some pre-chosen maximum) loops have been performed, at which time the data is written to file on the disk or returned to the calling function.

3.2 Potential Curve Decomposition Issues

The major issue one may encounter in Bertini_real when decomposing a curve is the failure to identify all critical points. Since the critical points include singular points of the curve, one needs to be willing to use higher, tighter, and more stringent thresholds when performing tracking using `Bertini` to be able to compute these with confidence. Experimentally, we have found the most important settings tend to be `SecurityMaxNorm`, `TrackTolBeforeEG`, and `ODEPredictor`, with `CondNumThreshold` being set higher to allow points closer to singular points to pass through the post-processor.

When one fails to find all the critical points, for whatever reason, the problem manifests in attempts to track midpoints upstairs across critical boundaries. These paths fail numerically, as the tracked point approaches and passes over a singularity. Consequently, Bertini_real takes longer

to run, and produces incorrect results. Furthermore, refinement of incorrect decompositions will fail as well.

4 Surface Decomposition

Imagine your favorite person. Almost certainly your mental image involves the person's outline, their silhouette. Your eyes naturally perform a sort of decomposition of the person, in terms of the boundary and their face. You can see where they begin and end, and if you were to draw them, you perhaps would start by drawing their outline. Bertini_real decomposes a surface in much the same way, by first computing the outline of the object from a certain projection.

Bertini_real implements the algorithm from [13], with modifications as necessary and improvements where available. The most notable difference is removal of the 'almost smooth' restriction of [13] by using the inclusion of deflation methods to handle singular curves.

4.1 Six Steps to Decompose an Algebraic Surface

As with the curve decomposition, the preliminary step of surface decomposition is to obtain a witness set for the surface being decomposed, which can be computed using Bertini. Working from a witness set allows us to focus on particular component(s) of interest, excluding others by virtue of the fact that we will not regenerate from them. Let S be the irreducible component to study.

The steps to decompose an algebraic surface are similar to those of the curve method. We spend much time regenerating from the given witness linear equations to specific systems in order to solve particular problems related to the surface, then build faces, the 2D analog to edges, from previously computed information.

1. Decompose the nonsingular critical curve.
2. Decompose the singular curves
3. Intersect with a sphere.
4. Slice at and between critical points.
5. Connect the dots, to form faces.
6. Refine the decomposition.

4.1.1 Critical Curve

The *critical curve* of S is the curve in ambient space over which the Jacobian is singular with respect to the two chosen projections, π_0, π_1. The two projections (that is, the plane which is formed by considering $(\pi_0(x), \pi_1(x))$ as a coordinate pair) will parameterize the surface implicitly, and when the surface is tangent to the projections, this is the critical curve. More formally, the critical curve of S with respect to π_0, π_1 is the set of solution components in S that satisfy

$$F(x) = \begin{bmatrix} f(x) \\ \det \begin{pmatrix} Jf(x) \\ J\pi_0 \\ J\pi_1 \end{pmatrix} \end{bmatrix} = 0. \tag{7}$$

By assumption, S is an irreducible component of $\mathcal{V}(f)$, but there may be other components in $\mathcal{V}(f)$ as well. We restrict all computations to S by using homotopy intersection methods that begin with a witness set for S. For more on intersection methods, see [32, 23, 25] or [12, Chap 12].

Critical curves can consist of multiple components, in the sense that there may be one part which is critical only with respect to the particular projections being used, and other parts which are critical independently of the choice of projection. The non-singular critical curve requires no deflation, and will manifest differently under varying projection, while those parts which appear regardless of projection are singular and require deflation for tracking.

Note that the inclusion of the ability to treat surfaces including singular curves is a significant improvement over the originally published algorithm, which provided only for surfaces with finitely many singularities.

Witness Set for the Critical Curve First, we need to obtain the witness points for the critical curve. We perform a linear product regeneration from the surface witness set to a nullspace-type system, as:

$$\begin{bmatrix} f(x) \\ v^{\intercal} \cdot \begin{bmatrix} Jf(x) \\ J\pi_0 \\ J\pi_1 \end{bmatrix} \\ L_1(x) \\ \text{patch}_v(v) \end{bmatrix} = 0 \tag{8}$$

Again, this is equivalent to finding $S \cap V(F(x)) \cap V(L_1(x))$ but replacing the determinantal condition in $F(x)$ with the existence of a null vector, v, instead. After finding solutions (x, v) to (8), we discard v and keep just the critical points.

Note the presence of the linear $L_1(x)$ in the system (8). This linear is included as a random complex linear slice, so as to give us general witness points on the *complex* critical curve. Hence, for the regeneration, we homotope only one of the two original witness linear equations to the starting linear equations in x, leaving the other fixed to become the random complex slicing linear for the critical curve witness set.

The critical curve system (8) will necessarily contain the singular curves on the surface, as they have a higher-dimensional tangent space. That is, in solving (8), we will pick up witness points for these singular components, as well as witness points for the well-behaved non-singular critical curve. Although both types of curves are critical, we will refer to the nonsingular critical curve as the critical curve, and disambiguate the rest by referring to them by the apparent multiplicity of the witness points and an integer index (there may be more than one singular curve of any given multiplicity).

Nonsingular Critical Curve We use π_0 to decompose all components of the critical curve, so we need to find the critical points with respect to π_0. The system which defines these points is

$$\begin{bmatrix} F(x) \\ \det \begin{pmatrix} JF(x) \\ J\pi_0 \end{pmatrix} \end{bmatrix} = 0. \tag{9}$$

This problem is, of course, the same as finding the critical points of a curve as in §3.1.1, except with $F(x)$ in place of $f(x)$. Hence, we solve it the same way as in that section, using a nullvector formulation to avoid the determinant and starting the intersection operation with the witness set for the critical curve as found above. Unfortunately, $F(x)$ already contains a determinant, which

we must differentiate to get $JF(x)$. The resulting complexity of evaluating the expression and its potentially high degree limit the size of problem that Bertini_real can handle.

The critical curve is not merged.

Singular Curves This portion of the paper along with [7] serve to complete the algorithm from [13] by removing the *almost smooth* condition. This condition required that the surface was smooth everywhere except possibly at finitely many points. In particular, the surface could not contain a singular curve such as the "handle" of the Whitney umbrella. The fundamental issue is the ability to effectively track along such singular curves and compute a cell decomposition.

The key to handling singular curves is to use isosingular deflation [24]. This approach allows one to construct a witness system for each singular curve thereby permitting a cell decomposition. In order to apply this technique, one first needs to compute a witness point set for the singular curves. The witness set for the critical curve computed above will contain a witness point set for both the critical curve and any singular curves the surface may have. One identifies the points on the singular curves based on the rank deficiency of Jf.

The set of witness points lying on singular curves is first partitioned based on the so-called deflation sequence [24] since witness points on the same curve must have the same deflation sequence. For each deflation sequence, one simply uses standard numerical irreducible decomposition techniques to decompose the set of corresponding points based on the common deflated polynomial system. This yields witness sets for the irreducible singular curves on the surface.

For each irreducible singular curve, we decompose the curve using the deflated system. We again do not merge.

4.1.2 Sphere Intersection Curve

To treat any non-compact portions of S, we intersect S with a sphere which is automatically computed and contains the critical points of the critical curves and each singular curve. Alternately, the user may supply any sphere which interests them.

To decompose any curve we require witness points, so the first step to decomposing the curve of intersection of S and the bounding sphere is to compute witness points. We regenerate from the two supplied witness linear equations for S, holding one fixed to serve as the slicing linear for this curve's witness set, and move the other to the two necessary start linear equations: we need two because intersection with a sphere is of degree 2. Thereafter we compute the critical points and slice as we would any other curve.

It is vital not to perform the merge operation on the sphere intersection curve.

4.1.3 Slicing

At this point we are ready to slice the surface. We know all the interesting topology happens at the critical points of the critical curve, singular curves, and sphere intersection curve, so we slice at and halfway between each consecutive pair of critical projection values. For brevity, we refer to the slices at critical projection values as *critslices* and the slices between critical projection values as *midslices*. Let χ be the union of all critical points of the aforementioned curves, and $\{p_{c_j}\} = \pi_0(\chi)$ be the set of projection values.

For each distinct p_{c_j} and midpoint $p_{m_i} = \frac{1}{2}(p_{c_i} + p_{c_i+1})$, perform a curve decomposition of the intersection of S with the appropriate linear slice, either $(\pi_0(x) - p_{c_j})$ or $(\pi_0(x) - p_{m_i})$. We use the second projection $\pi_1(x)$ to decompose each slice. All of the slice's critical points will lie on the critical curve, a singular curve, or the sphere and will appear as previously, as a computed and

noted point in the running collection of vertices. The midpoints of edges on the midslices (the slices at the p_{m_j}) will become the midpoints for the faces of the cellular decomposition, and will need to be 'glued' to the midpoints of the critslices (the slices at the p_{c_j}) in the connect-the-dots phase.

4.1.4 Connecting the Dots

This is perhaps the most subtle of the steps. Up to this point, we essentially have a relatively sparse cloud of curves on the surface, and we wish to glue the edges together preserving topological information. To do so, we will simultaneously track on three systems at once, coupling them together via functions regarding the solutions' projection values.

One cannot simply perform a straight line homotopy from the midpoint of a face to a target pair of projection values – the face may not be convex with respect to $(\pi_0(x), \pi_1(x))$! Hence, if one tried to follow a straight line in terms of projection, they very well might cross a singularity. This is the very reason we work so hard to find the critical and singular curves – they are the boundaries we cannot cross when tracking in terms of projection value. To the left and right of the faces, we have explicit bounds in terms of π_0, which are formed by the critical slices. In contrast, the bounds for π_1 are implicitly formed by the π_1 values of the top and bottom edges coming from the critical curve, and you must do something special to ensure that you do not inadvertently cross its boundary.

A very special homotopy is presented in (10) which ensures that we track away from boundaries between cells and therefore never cross any singularities. Essentially, there are three systems we track simultaneously – the face of the surface itself, and the top and bottom boundary curves. Additionally there are three functions which force the point on these three curves to take on the correct π_0 value. Finally, the last function forces the point in the middle of the face to move in terms of π_1, staying between the two boundary points.

We note here that there is a typo in the paper [13] in which the algorithm for this decomposition appeared – the denominator in the final function should be absent. The correct homotopy is

$$\begin{bmatrix} f(x) \\ \pi_0(x) - [(1-u)\pi_0(c_i) + u\pi_0(c_{i+1})] \\ f_{\text{bottom}}(y) \\ \pi_0(y) - [(1-u)\pi_0(c_i) + u\pi_0(c_{i+1})] \\ f_{\text{top}}(z) \\ \pi_0(z) - [(1-u)\pi_0(c_i) + u\pi_0(c_{i+1})] \\ \pi_1(x) - [(1-v)\pi_1(y) + v\pi_1(z)] \end{bmatrix} = 0. \tag{10}$$

The homotopy is slightly hidden in (10), in that as written there is no explicit dependence on t. To track the homotopy, we use

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} (1-t)\, u_{\text{target}} + t\, u_{\text{start}} \\ (1-t)\, v_{\text{target}} + t\, v_{\text{start}} \end{bmatrix}. \tag{11}$$

So long as we do not venture outside the unit box $(u, v) \in [0, 1]^2$, the homotopy defined in (10) is nonsingular. There may be singularities on the boundary, but we always track from the interior to the boundary, so this is no barrier to computation since Bertini's tracker uses singular endgame methods to approximate singular solutions.

Using the special homotopy (10) in Algorithm 1, we iterate over each edge in each midslice, connecting its midpoint to all appropriate midpoints of edges in the lower and upper critslices, as in Figure 7. There may be multiple critslice edges to which a midedge maps, and we must find

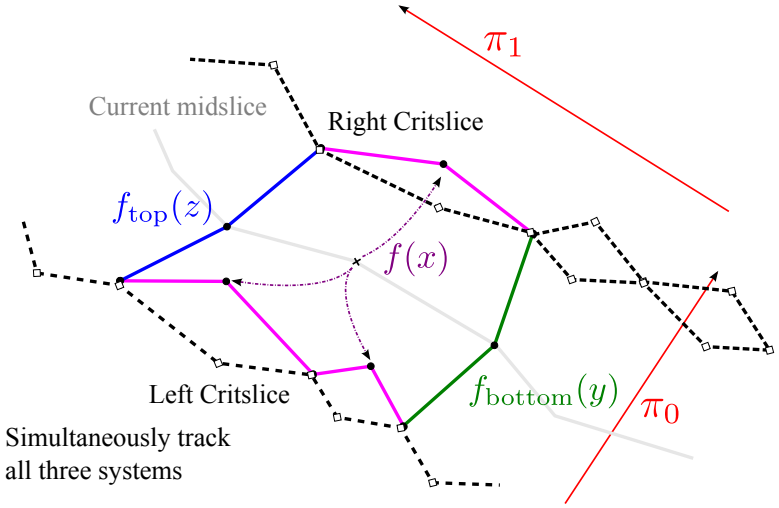

Figure 7: The midtrack system (10) is employed by Bertini_real to connect midslice edges to critslice edges. Simultaneously track each of f, f_{bottom}, and f_{top}, together with four equations coupling their π_0 and π_1 projection values. The use of this special homotopy guarantees that the midpoint stays away from the bounding edges, except at the end of the homotopy.

them all. The implemented method works from the lower end of the top and bottom slices, finding all possible edges, and tracking to find the single edge which connects next.

There are a few subtleties regarding Algorithm 1. Foremost is the fact that in order for it to function properly, the sub-decomposition curves for input surface S must have been merged appropriately. More specifically, the slices, both mid and critical, must be merged, so that there are no 'new' points as endpoints of edges. This is because the endpoints **must** lie on the critical, singular, or sphere curve. Otherwise, the new point is in the interior of a face, and there is no corresponding system on which to track! Correspondingly, the critical curves should *not* be merged. Otherwise, there will be mid or critslice edges which do not properly terminate on the critical curve, again preventing ConnectTheDots from running correctly.

Another fine point is that when the midtrack system terminates, the point it finds might be found as a removed point on one of the critslice edges. That is, when the merge process happens in curve decomposition, we cannot simply forget about the 'new' points which are merged away. Rather, we must record that these points do in fact lie on their respective edges. Otherwise, we may end up tracking to them, but no longer having a reference, we are unable to make the connection.

ConnectTheDots marks the end of the central surface decomposition algorithm.

4.1.5 Refinement

Refinement of a surface decomposition is relatively straightforward, although the process of programming a more optimal refinement strategy is ongoing. At this time, a naive two-step refinement procedure is implemented as briefly follows.

The first step is to refine each [nondegenerate] edge of each curve, including the critical, singular, and sphere intersection curves, and all mid- and critslices, so that each has a given number of points. Secondly, we add points to each face, by forming a rib of samples which all lie on over a single π_0 value. Finally, we connect the face ribs to the edges to form a new triangulation.

Algorithm 1 Connect midslice edge to critslices

```
1: function ConnectTheDots(surface & S)
2:     for each midslice of S do
3:         for each edge of current mislice do
4:             MakeFace(S, slice index, edge index, F)
5:         end for
6:     end for
7: end function

1: function MakeFace(surface S, index ii, index jj, face & F)
2:     Set start point from midslice ii, edge jj
3:     Determine top and bottom systems and edges
4:     for Left and Right do
5:         Initialize current upper index as endpoint of bottom edge
6:         Set u_target = 0 if left, else u_target = 1
7:         while upper index != final index do
8:             Determine set of possible matching critslice edges given current upper index
9:             for each possibility, until have match do
10:                Set target v_target value, from current test edge's midpoint
11:                Homotope system (10) from (0.5, 0.5) to (u_target,v_target)
12:                Find edge with midpoint == returned point
13:                Update current upper index as right point of found edge
14:            end for
15:        end while
16:    end for
17: end function
```

4.2 Potential Surface Issues

Surface decompositions in Bertini_real are subject to the same issues as curves. The finding of critical points for all involved curve decompositions is paramount, as the loss of any causes edge creation problems. Numerical settings in `Bertini` can be chosen suitably to minimize these issues.

One larger issue for surfaces is the determinant for the critical curve. The critical curve defined by (8) requires the determinant of the Jacobian (along with the Jacobian of the projections) to be 0. Finding witness points for the curve is accomplished by using a null space approach to avoid taking the determinant. However, computing critical points of the critical curve is the major barrier, as we currently lack the techniques to write the system down in a fashion that avoids taking determinants again. This is the subject of further research, and the removal of this barrier will allow Bertini_real to practically decompose surfaces in much higher dimension.

5 Implementation

5.1 Language choice, compilation, & library linking

Bertini_real is implemented in C++ and compiles against a number of libraries. It has been developed and tested in Linux and OSX but has not been tested in Cygwin.

The libraries against which Bertini_real compiles are:

- MPFR,
- GMP,
- Bertini, suitably compiled into a library,
- Boost, most namely Boost.Filesystem, but including others,
- MPI.

Bertini_real is built from source using standard methods, and requires the C++11 standard. The source code is available to download from an online repository.

License Bertini_real is released under a Bertini-style license, with free-to-read source code available from a repository, but including a non-distribution clause. At such time Bertini changes its license to be more permissive, Bertini_real's license will also change. The license is available at the download page.

5.2 Major Data Types

Bertini_real provides its own data types for the primary objects – decompositions – as well as some which extend the functionality of Bertini. Since Bertini is written in C and relies on frequent explicit initialization and clearing of variables, many of Bertini_real's data types lie on top of those from Bertini. We briefly describe the two most important new data types, leaving the rest of the details to the documentation.

Decompositions The curve and surface decomposition are subclasses of the `decomposition` class, which stores the dimension, component number, projections used, the name of the input file, number of variables, and the generating witness set. The specific curve and surface classes contain edges and faces, respectively. Surfaces contain vectors of curve decompositions for the critslices and midslices, as well as nonsingular critical curve, singular curves, and sphere intersection curve.

Vertex Set As Bertini_real runs, the program accumulates points into a common database, for retrieval by index at a later time. The `vertex_set` class is Bertini_real's way of doing so; provides an interface for finding the index of a given point, as well as storage of metadata such as its type, in which system is was originally found, and its projection values.

5.3 Quirks and Subtleties

Bertini_real uses Bertini's tracker loop as its main computational engine. In order to be able to provide off-the-shelf decomposition of affine varieties, there is required to be a single `variable_group`, and the program will automatically add a homogenizing variable and work over projective space with a random complex patch. Since all computations require a patch, Bertini_real does not dehomogenize any solutions for storage in the `vertex_set`, as dehomogenization at storage would require re-finding a patch every time a point was to be used as a start point for a homotopy. Consequently, when the output data is written to disk, all points are written without dehomogenization, and without any scaling. That is, in order to view the decomposition, the user must dehomogenize the points.

Another further note is that some points are found via nullspace methods. These points have additional coordinates corresponding to the auxiliary variables, and will appear in the `vertex_set`

with them. During sampling the decomposition must be reloaded, and these extra variables allow us to skip re-computing the auxiliary variables. It is a simple enough matter to discard the extra variables at process time.

5.4 Parallelism

One of the current limitations for parallelism is that the `vertex_set` must be maintained by the head MPI process. Since the decomposition data types currently in use have been implemented using integer indices into the `vertex_set`, if two different processes label points differently, or have the same index refer to different points, the data will desynchronize and become invalid. This problem is the subject of ongoing improvement, and could be solved using smart pointers, for example. Nonetheless, we here briefly describe the existing parallelism. Bertini_real is an MPI-parallel program. Parallelism comes in two distinctly different forms.

While tracking multiple paths When multiple start points are to be tracked for a given system, with the same parameters, we use Bertini-style parallelism, where we send start points in groups to the worker processes, the workers track to solutions, and they report back to the master.

During ConnectTheDots During Algorithm 1, the only new data being created are the faces, and this process does not require the addition of points to the `vertex_set`. This means that we can maintain synchronousness of the `vertex_set` while performing the computations in parallel.

Additionally, the ConnectTheDots() routine tracks only a single path for each tracker call, meaning that tracker-level parallelism is impossible. Therefore, we implemented a face-construction level parallelism into the for loops of ConnectTheDots(). In the inside loop, the master sends the two indices of the face to be created, the worker calls MakeFace(), and the sends the master the finished face.

5.5 Input and output formats

Input and output are through plain-text files written to disk.

5.5.1 Input

The necessary input files for Bertini_real are:

1. The same Bertini `input` file from which the user acquires a numerical irreducible decomposition;

2. The `witness_data` file created by Bertini after computing a numerical irreducible decomposition.

Many systems have multiple components. Bertini_real therefore processes `witness_data` for each component, and offers the user a choice if there is one. That is, if there is a single component of dimension one or two, Bertini_real will automatically and implicitly decompose that component. If there are multiple components, Bertini_real will prompt the user for their choice. They may only decompose a single dimension at a time, but they may elect to decompose as many components as they wish simultaneously, provided that they all have the same deflation sequence. For example, the system in §6.1.1 has multiple components of dimension one, some of which are nonsingular, and several of which must be deflated. The images in this example were produced by decomposing several components at the same time.

Optionally, the user may supply:

- a sphere file, consisting of the radius and center; and/or
- a projection file, prefaced by the number of variables, and containing the coefficients of the linear projections π_i.

The user indicates the names of the files using flags to the command line; *e.g.* calling `bertini_real -sphere mysphere -pi myprojection`.

5.5.2 Output

There are two kinds of output: 1) temporary files generated by the Bertini parser and produced input files, and 2) the end-result output deposited into a subfolder of the current working directory. The temporaries are unavoidable but inconsequential, so they are not discussed here. The writing of files is incremental, and happens after each major stage, so that if the program fails for some reason, the user gets the most recent good state as a parsable set of data.

Files common to all decompositions

- `vertex_set`
  The number of vertices, the number of projections, the number of natural variables, and the number of input files referenced. Coefficients of the π_i projections. The input file names. Then the vertices, appearing as the number of variables, the coordinates, the number of projection values stored, the projection values, and the integer index of the filename for which the point was originally found and committed.
- `input` file
  Copied verbatim into the folder. Needed for sampling and future reference.
- original `witness_data` file
  Copied verbatim into the folder. This file is part of the generating data, and so is deemed part of the decomposition.
- `decomp` file
  The name of the input file, the number of variables, and the dimension of the decomposition. All of the π_i projections used. The patches. The center and radius of the sphere.

Files specific to curve decompositions

- `E.edge`
  Contains all edges of the curve, listed as integers into `vertex_set`, being line-delimited as `left mid right`.

Files specific to surface decompositions

- `S.surf`
  The number of faces, the number of critical slices, and the number of midslices, along with the number of singular curves, their 'multiplicities', and the number of each multiplicity.
- All curve sub-decompositions, written to their own sub-folders
  critical curve, sphere curve, singular curves, and all mid and critslices.

Full documentation for output formats is detailed using Doxygen, and is available at the Bertini_real webpage.

5.6 Visualization

In Matlab Visualization for Bertini_real is currently provided through Matlab, using object-oriented code, as in Figure 8, which shows the 'calyx' example of [26]. This is a two-step process: The data are gathered, then plotted. Plotting the data creates a handle class object in the workspace, which the code uses to manipulate the scene, but interactivity is achieved with buttons and tickboxes rendered into the screen. At call time, there are options to limit the amount of data rendered. For example, complicated surfaces may have thousands of edges and tens of thousands of points; in this case, the user may with to omit the drawing of embedded curves and vertices, as well as their labels. If the user wants to color the surface monochromatically rather than with the default jet scheme, they may do that as well. The UI automatically saves an image of the scene upon drawing, as well.

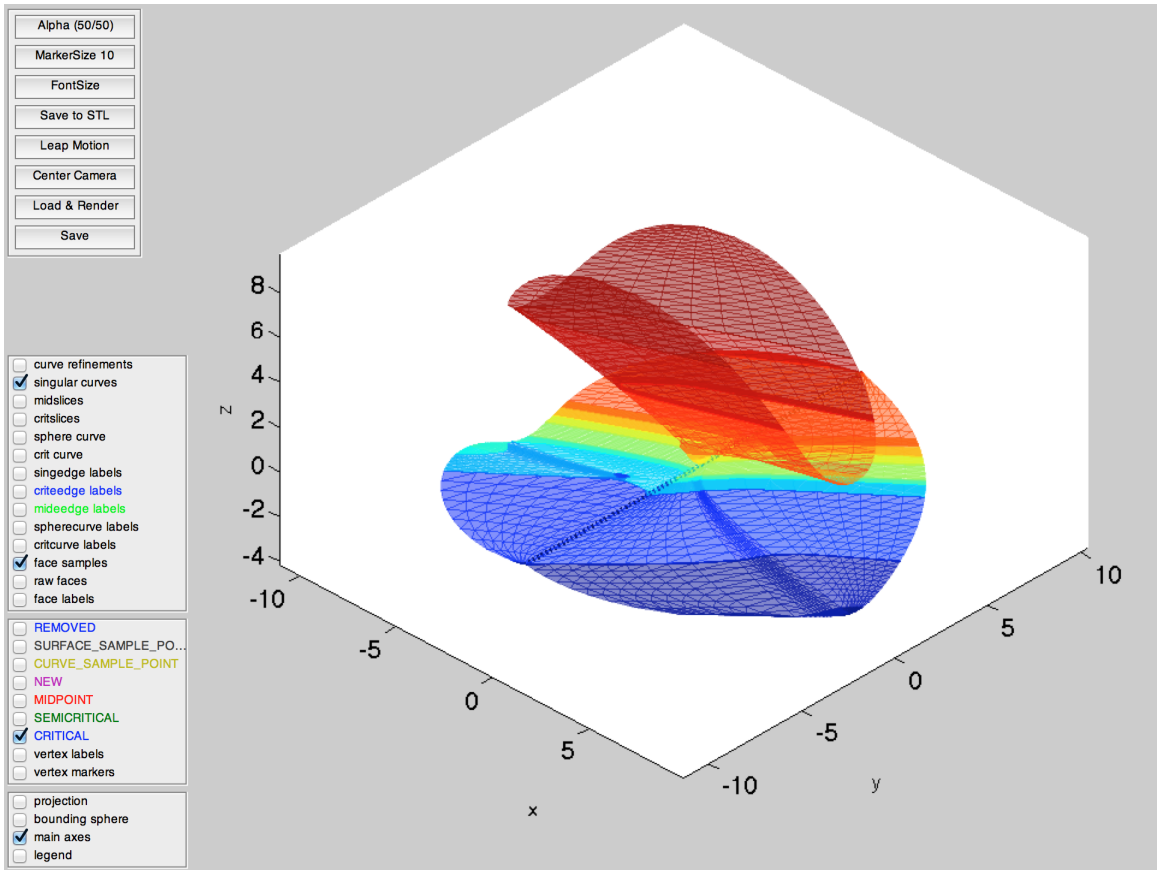

Figure 8: Bertini_real's default surface visualization UI in Matlab. By default, each face gets its own color, and all curves and vertices are rendered with tickboxes to turn on and off labeling. Saving is done with the push of a button, and there are options to write the triangulation to an STL file and to control the figure with a Leap Motion controller.

If the system has more than three variables, the Matlab code will look for coordinates which are constant on the decomposition, and offer the user a choice of non-constant coordinates onto which

to project the decomposition. If the user wants a more complicated projection for visualization or data processing, they may pass a handle to the constructor of the object at call time, and all points will be passed through the function before rendering. For an example of post-decomposition projecting, see Section 6.2.2.

3D Printing Three-dimensional printing of surfaces decomposed using Bertini_real is almost trivial. Since even the unrefined surface decomposition is a triangulation, 3D printing is as simple as:

1. Convert data from plaintext,
2. Write to stereolithography (STL) file,
3. If not compact, solidify the surface,
4. Process in model repair service,
5. Graphically ensure quality, and
6. Print.

We have been successful in printing surfaces, both compact and unbounded, those which are everywhere smooth, those which contain cusp singularity points, and even those containing singular curves.

One of the main difficulties with 3D printing a surface is to give it positive measure. Unless the surface is totally bounded, or the data has been processed through a special projection, at least part of the surface will have zero measure inside $\mathbb{R}^3$, and the 3D printing software, which slices parallel to the printing plane, will error, producing missing traces or areas to fully fill. To solve the measure problem, we have had reasonable success with the 'Solidify' routine, available in Blender [20]. For example, Figure 9 depicts the printing of the 'Solitude' surface [26], and the Barth Sextic [37].

At singularities, the current solidify process often produces strange behavior, such as intersecting pieces, flipped normal vectors, and other artifacts. For example, at the cusp singularity in 'vis-a-vis' [26], the sharpness of the cusp is lost, and instead we have a cone-like object, with flipped normals. On the internal part of the handle of the Whitney Umbrella, since there is not necessarily a 'correct' direction for the normal vector for each half prior to thickening, centered thickening will produce symmetric pieces. This is in contrast to thickening only in the direction of the normal vector, which results in offsets.

Figure 9: Photographs of 3D prints of Bertini_real decompositions. Left: Solitude. Right: the Barth Sextic. Both from a U-Print using dissolvable support material.

Another problem with node singularities is that a surface can approach one from multiple sides at once. This is challenging for the printer, because the layers do not adhere perfectly, and the traces are very small. Hence, nodes for which printer paths pass through the singularity with little length will produce weak joints. In Figure 9, the Barth Sextic lost several of its 'cones', namely those at the top and bottom, given the orientation in which is was printed.

There are sufficiently many issues and subtleties with the 3D printing of non-smooth surfaces that further research on ways to thicken while respecting singularities is merited.

6 Examples

In this section, we provide illustrative examples for Bertini_real's application for curves and surfaces.

6.1 Curves

6.1.1 Foldable Griffis-Duffy

This is Ex. 4.4 from the original paper on decomposing real curves using numerical algebraic geometry [28]. A type of Stewart-Gough platform robot, the Griffis-Duffy mechanism has several special cases with particularly interesting motion studied in [27, 31]. The particular example considered here is a Griffis-Duffy Type II mechanism further specialized with leg lengths equal to the altitude of the platform triangles. This system of six homogeneous quadratics defined on $\mathbb{P}^7$ has a curve that factors irreducibly into 12 nonphysical lines, 3 double lines, 3 quadrics, and 4 quartics. Figure 10 reproduces the result of [28] (without the linear equations), using the same projection onto a plane. Whereas the original figure was a proof of concept programmed in MATLAB with some manual steps, Bertini_real completely automates the process. The decomposition took 175 seconds on a 2.6 GHz Intel Core i7 processor, using a single core.

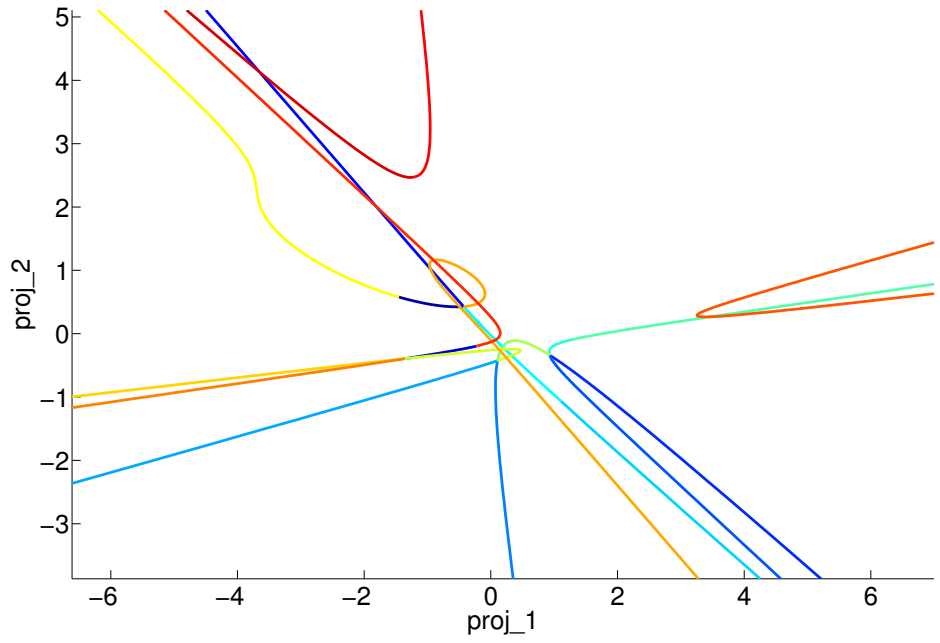

Figure 10: The six-bar Griffis-Duffy example. In this figure, we have used Bertini_real to decompose 3 quadrics and 4 quartics simultaneously. The double-linear equations have not been rendered.

6.1.2 Mixed Burmester Curves

Depending upon the needs of an application, a four-bar linkage can be used for "body guidance," where its coupler link undergoes a one-degree-of-freedom planar motion that coordinates its position and orientation (i.e., a curve in $SO(2)$), or it can be used to guide the path of a single point of the coupler link, so called "path synthesis." In 1888, Burmester [16] posed and solved the body guidance problems of interpolating 4 and 5 points in $SO(2)$ (see [14]), whereas the path synthesis problem of interpolating 9 points in the plane, which was posed by Alt in 1923 [2], was not completely solved until 1992 [36]. More recently, Tong [35] posed a generalization where the objective is to interpolate m points in $SO(2)$, where the orientation of the coupler link matters, and n points in $\mathbb{R}^2$, where orientation of the coupler link is ignored. We call these "mixed Burmester problems." The dimension and degree of the solution set depends on the values of m and n. For points in general position, an (m, n) mixed Burmester problem has a solution set of dimension $10 - 2m - n$, with the original Burmester problems being cases $(4, 0)$ and $(5, 0)$ and Alt's problem equivalent to case $(1, 8)$.

The $(3, 3)$ mixed Burmester problem has a solution curve, and in a formulation consisting of 14 polynomials in 13 variables, it has complex degree 630 with a critical set of size 228. For a particular choice of the precision data and projections, Bertini_real produces a decomposition containing 172 edges, unmerged. Figure 11 shows a projection of this curve onto the motion plane of the mechanism. In this projection, there are three nodes where the curve crosses itself many times, but curiously, this is an artifact of the special projection, as the pre-images of these points in the full ambient space have no crossings. The nodes have a simple kinematical relation to the three given points in $SO(2)$, an unexpected result revealed by this computation. This computation took about 12.6 hours on 65 AMD Opteron 6278 processors running at 2.4 GHz.

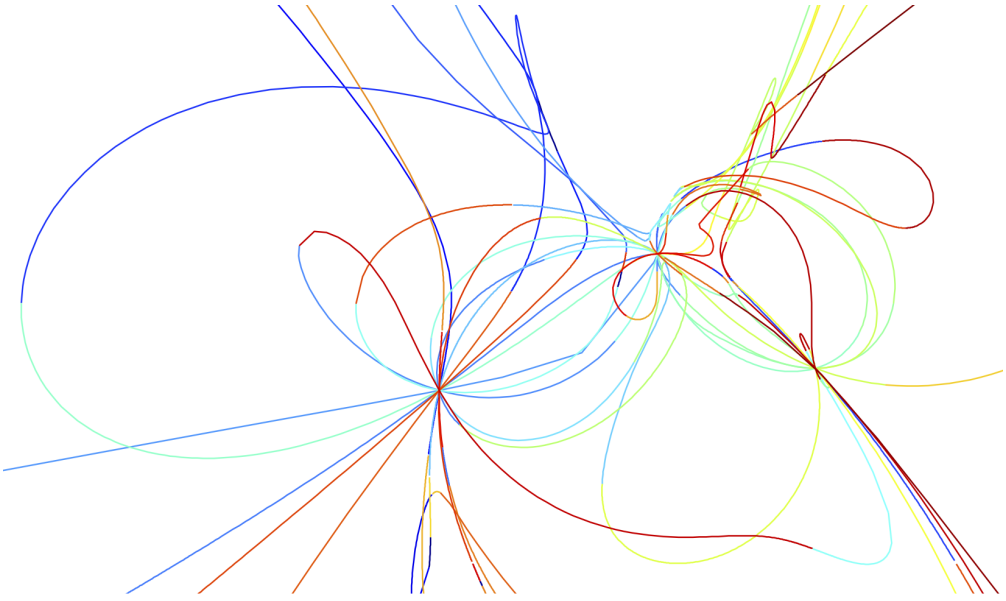

Figure 11: Burmester curve 3-3, of complex degree 630. The defining system consists of 14 equations in 13 variables. The colors of the curve indicate separate edges. While there are three apparent nodes, in the full ambient 14-dimensional space, there are no crossings there!

6.2 Surfaces

6.2.1 Bottle Opener

The original paper [13] on which our surface decomposition approach is based contains the following illustrative example. Defined by a single polynomial equation in three variables, namely

$$((x+0.35)^2(1-x^2)-y^2)^2+z^2-0.00531441=0, \tag{12}$$

the real solution set resembles a torus but with a pinch point at exactly $(x,y,z)=(-0.8,0,0)$. Figure 12 shows a rendering of this surface produced by Bertini_real. The surface is degree 4, and its critical curve is degree 20 with 5 critical points in $\mathbb{C}^3$. For the particular random real projection chosen for the decomposition, Bertini_real produced 10 faces, as color-coded in the figure. Runtime using 17 cores on a small cluster, with AMD Opteron 6278 processors at 2.4 GHz, was approximately 95 minutes. Most of the runtime occurs when finding the critical points of the critical curve, and is largely due to having polynomials of high degree. Removal of the determinantal calculation would improve runtime. Currently implemented parallelization is far from optimal as well.

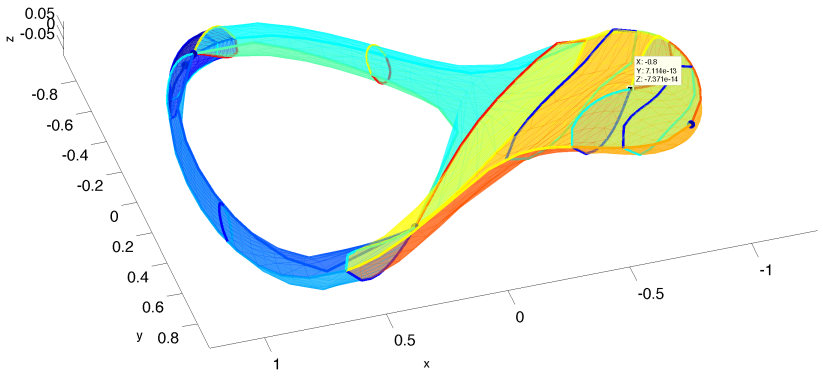

Figure 12: The bottle-opener surface, with a hole and a pinch point, used as the illustrative example from [13]. Decomposed here using Bertini_real with respect to random projections.

6.2.2 Pentagonal Linkage

Consider a planar linkage consisting of five rigid links, each of unit length, hinged in a loop. Holding one link stationary, the remaining links can move with two degrees of freedom; that is, one can rotate two of the hinges arbitrarily (within certain limits) but these then determine the other three hinge angles. In short, we have a deformable regular pentagon whose set of possible configurations forms a surface. Number the links in order around the loop and let $s_i=\sin\theta_i$ and $c_i=\cos\theta_i$, $i=0,\ldots,4$, where θ_i is the absolute rotation of the i-th link in the plane. We assume link 0 is fixed, with $\theta_0=0$. The closure condition for the linkage is

$$(s_1+s_2+s_3)^2+(1+c_1+c_2+c_3)^2=1, \tag{13}$$

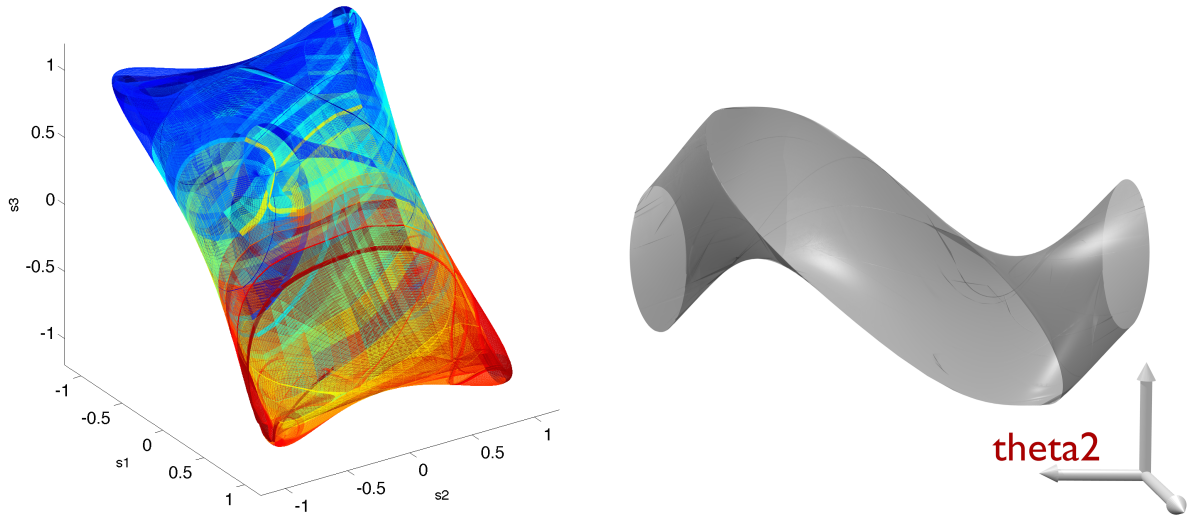

Figure 13: The fivebar mechanism surface, presented in two projections. We show on the left the projection onto (s_1, s_2, s_3), and on the right, the projection onto (θ_2, x, y).

and the sine-cosine pairs must satisfy

$$s_i^2 + c_i^2 = 1, \qquad i = 1, 2, 3. \tag{14}$$

Hence, we have 4 quadratic equations in 6 variables.

Figure 13 shows images derived from the real solution set of this system. On the left is a rendering of the projection of the set onto (s_1, s_2, s_3). Although in this view only four protrusions are visible, the surface actually has six. A picture with a more intuitive interpretation can be developed as follows. Consider tracking the movement of the apex of the pentagon, i.e., the point where link 2 connects to link 3. Selecting the stationary end of link 1 as the origin and the direction of link 0 as the x-axis, the coordinates of the tracking point are

$$(x, y) = (c_1 + c_2, s_1 + s_2). \tag{15}$$

The right image in Figure 13 shows a projection of the motion surface to (θ_2, x, y). (We show just the main branch of $\theta_2 = \text{atan2}(s_2, c_2)$; the surface may be considered an infinite chain joining end-to-end identical copies of the clipped surface.) The surface is a kind of twisted tube, where for some values of θ_2, the tracking point makes a full circle in (x, y), while for others, only a partial circle is obtained. What looks like a sharp edge around the missing part of the tube is not a singularity in the full solution set; it is a figment of the projection. The smooth surface in the ambient space double covers the twisted tube in a manner such that it folds onto itself around the sharp edge of the hole.

Computation time for this decomposition was approximately 15.3 minutes using 65 AMD Opteron 6278 2.4 GHz cores. Sampling to a level of 20 points per edge took about 10.4 hours using a single processor, as sampling is not currently parallelized.

7 Conclusion

Bertini_real is a fully automated implementation of the proof-of-concept ideas presented in [28] and [13] for cell decomposition of real algebraic curves and surfaces. It is built on top of the capabilities of Bertini [8] for numerical irreducible decomposition in complex space and for homotopy path tracking. In principle, the methods work for curves and surfaces inside ambient spaces of any dimension, but in its current form, computational cost restricts the dimension of the ambient space to about 20 for curves and 8 for surfaces. Additionally, Bertini_real implements isosingular deflation [24] to handle sets of multiplicity greater than 1. This can occur in the original curve or surface to be decomposed, or can crop up as a singular curve within a surface of multiplicity 1.

Our approach solves for all critical points with high precision. This means that the method resolves the true topology of the set at this precision without finely subdividing the entire curve or surface, as might be done in an exclusion method. Since Bertini has multiprecision capability, the precision of the topological analysis can go well beyond double precision, although at a cost in computation time. It must be said, however, that computation of the critical points can involve solving polynomial systems of high degree and this can be a difficult numerical task. See Sections 3.2 and 4.2 for further discussion. Besides computing cell decompositions, the package contains facilities for triangulating the cells to finer resolution and displaying projections of the triangulations in a 3-D graphical environment.

Bertini_real has opportunities for further improvement. The isosingular deflation currently relies on the symbolic toolbox in MATLAB, as does the creation of critical curve input files, which induces a software dependency, which would be nice to remove. More importantly, we currently use a determinant in the equation used to solve for the critical points of the critical curve of a surface. The inefficiency of evaluating determinants is the major roadblock to computations in higher ambient spaces. Compactification of noncompact sets happens by intersecting them with the interior of a bounding sphere. Some mathematical investigations might prefer to compactify in a different manner. For example, in [13], a method is described for compactifying surfaces in real projective space so that the cell decomposition effectively extends to infinity.

The generalization of real cell decomposition to sets of higher dimension would be valuable. An extension to three-folds would be particularly useful in engineering applications, such as for mapping robot workspaces. The natural generalization would begin by finding the critical surface of the three-fold with respect to a projection onto a three-plane. One sticking point will be that this surface will need to be decomposed into cells, and hence one will ultimately need to find the critical points of the critical curve of this critical surface. Formulating this using nested determinants would no doubt lead to intractable complexity for all but the simplest of three-folds. Avoiding this complexity is a topic of ongoing research.

References

[1] L. Alberti, G. Comte, and B. Mourrain. Meshing implicit algebraic surfaces: the smooth case. *Mathematical Methods for Curves and Surfaces: Tromso*, 4:11–26, 2004.

[2] H. Alt. Über die Erzeugung gegebener ebener Kurven mit Hilfe des Gelenkvierecks. *Zeitschrift für Angewandte Mathematik und Mechanik*, 3(1):13–19, 1923.

[3] A. Appel. Some techniques for shading machine renderings of solids. In *Proceedings of the April 30–May 2, 1968, Spring Joint Computer Conference*, AFIPS '68 (Spring), pages 37–45, New York, NY, USA, 1968. ACM.

[4] D.S. Arnon, G.E. Collins, and S. McCallum. Cylindrical algebraic decomposition I: The basic algorithm. *SIAM Journal on Computing*, 13(4):865–877, 1984.

[5] S. Basu, R. Pollack, and M.-F. Roy. *Algorithms in Real Algebraic Geometry (Algorithms and Computation in Mathematics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[6] D.J. Bates, D.A. Brake, J.D. Hauenstein, A.J. Sommese, and C.W. Wampler. Homotopies for connected components applied to computing critical sets, 2014.

[7] D.J. Bates, D.A. Brake, J.D. Hauenstein, A.J. Sommese, and C.W. Wampler. On computing a cell decomposition of a real surface containing infinitely many singularities. In Hoon Hong and Chee Yap, editors, *Mathematical Software ICMS 2014*, volume 8592 of *Lecture Notes in Computer Science*, pages 246–252. Springer, 2014.

[8] D.J. Bates, J. D. Hauenstein, A.J. Sommese, and C.W. Wampler. Bertini: Software for numerical algebraic geometry, 2006.

[9] 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. In *Approximate commutative algebra*, Texts Monogr. Symbol. Comput., pages 55–77. SpringerWienNewYork, Vienna, 2009.

[10] D.J. Bates, J.D. Hauenstein, C. Peterson, and A.J. Sommese. A numerical local dimensions test for points on the solution set of a system of polynomial equations. *SIAM J. Numer. Anal.*, 47(5):3608–3623, 2009.

[11] D.J. Bates, J.D. Hauenstein, and A.J. Sommese. Efficient path tracking methods. *Numerical Algorithms*, 58(4):451–459, 2011.

[12] D.J. Bates, J.D. Hauenstein, A.J. Sommese, and C.W. Wampler. *Numerically solving polynomial systems with Bertini*, volume 25. SIAM, 2013.

[13] G.M. Besana, S. Di Rocco, J.D. Hauenstein, A.J. Sommese, and C.W. Wampler. Cell decomposition of almost smooth real algebraic surfaces. *Numerical Algorithms*, 63(4):645–678, 2013.

[14] O. Bottema and B. Roth. *Theoretical Kinematics*, volume 24 of *North-Holland Series in Applied Mathematics and Mechanics*. North-Holland Publishing Co., Amsterdam, 1979. Reprinted by Dover Publications, New York, 1990.

[15] D.A. Brake, D.J. Bates, W. Hao, J.D. Hauenstein, A.J. Sommese, and C.W. Wampler. Bertini_real: Software for one- and two-dimensional real algebraic sets. In Hoon Hong and Chee Yap, editors, *Mathematical Software ICMS 2014*, volume 8592 of *Lecture Notes in Computer Science*, pages 175–182. Springer, 2014.

[16] L.E.H. Burmester. *Lehrbuch der Kinematik*. Leipzig A. Felix, 1888.

[17] L.P. Chew. Guaranteed-quality mesh generation for curved surfaces. In *Proceedings of the ninth annual symposium on Computational geometry*, pages 274–280. ACM, 1993.

[18] G.E. Collins. Quantifier elimination for real closed fields by cylindrical algebraic decomposition. *Lec. Notes Comp. Sci*, 33:134–183, 1975.

[19] T. K. Dey and T. Ray. Polygonal surface remeshing with delaunay refinement. *Engineering with Computers*, 26(3):289–301, 2010.

[20] The Blender Foundation. Blender, 2014.

[21] A.S. Glassner. *An introduction to ray tracing*. Morgan Kaufmann, 1989.

[22] J.D. Hauenstein, A.J. Sommese, and C.W. Wampler. Regeneration homotopies for solving systems of polynomials. *Mathematics of Computation*, 80(273):345–377, 2011.

[23] J.D. Hauenstein and C.W. Wampler. Numerically intersecting algebraic varieties via witness sets. *Appl. Math. Comput.*, 2012. DOI:10.1016/j.amc.2012.06.034.

[24] J.D. Hauenstein and C.W. Wampler. Isosingular sets and deflation. *Foundations of Computational Mathematics*, 13(3):371–403, 2013.

[25] J.D. Hauenstein and C.W. Wampler. Numerical algebraic intersection using regeneration, 2014. Under review. Available at www.nd.edu/ jdhauens/preprints/hwGeneralIntersection.pdf.

[26] H. Hauser and J. Schicho. Algebraic surfaces, 2014.

[27] M.L. Husty and A. Karger. Self-motions of Griffis-Duffy type parallel manipulators. In *Proceedings of the 2000 IEEE Int. Conf. Robotics and Automation, CDROM, San Francisco, CA, April 24–28, 2000*. IEEE, 2000.

[28] Y. Lu, D.J. Bates, A.J. Sommese, and C.W. Wampler. Finding all real points of a complex curve. *Contemporary Mathematics*, 448:183–205, 2007.

[29] A. Paiva, H. Lopes, T. Lewiner, and L.H. de Figueiredo. Robust adaptive meshes for implicit surfaces. In *Sibgrapi 2006 (XIX Brazilian Symposium on Computer Graphics and Image Processing)*, pages 205–212, Manaus, AM, october 2006. IEEE.

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

[31] A.J. Sommese, J. Verschelde, and C.W. Wampler. Advances in polynomial continuation for solving problems in kinematics. *ASME J. Mech. Design*, 126(2):262–268, 2004.

[32] A.J. Sommese, J. Verschelde, and C.W. Wampler. Homotopies for intersecting solution components of polynomial systems. *SIAM J. Numer. Anal.*, 42(4):1552–1571, 2004.

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

[34] The CGAL Project. *CGAL User and Reference Manual*. CGAL Editorial Board, 4.4 edition, 2014.

[35] Y. Tong. Four-bar linkage synthesis for a combination of motion and path-point generation, 2013.

[36] C.W. Wampler, A.P. Morgan, and A.J. Sommese. Complete solution of the nine-point path synthesis problem for four-bar linkages. *ASME J. Mech. Design*, 114:153–159, 1992.

[37] E.W. Weisstein. Barth sextic, 2014.