Bertini_real: Software for One- and Two-Dimensional Real Algebraic Sets

Daniel A. Brake[1], Daniel J. Bates[2], Wenrui Hao[3], Jonathan D. Hauenstein[4], Andrew J. Sommese[5], and Charles W. Wampler[6]

[1] North Carolina State University, USA
danielthebrake@gmail.com,
danielthebrake.org
[2] Colorado State University, USA
bates@math.colostate.edu,
www.math.colostate.edu/∼bates
[3] Mathematical Biosciences Institute, USA
hao.50@osu.edu
people.mbi.ohio-state.edu/hao.50
[4] North Carolina State University, USA
hauenstein@ncsu.edu,
www.math.ncsu.edu/∼jdhauens
[5] University of Notre Dame, USA
sommese@nd.edu,
www.nd.edu/∼sommese
[6] General Motors Research and Development, USA
charles.w.wampler@gm.com,
www.nd.edu/∼cwample1

Abstract. Bertini_real is a command line program for numerically decomposing the real portion of a one- or two-dimensional complex irreducible algebraic set in any reasonable number of variables. Using numerical homotopy continuation to solve a series of polynomial systems via regeneration from a witness set, a set of real vertices is computed, along with connection information and associated homotopy functions. The challenge of embedded singular curves is overcome using isosingular deflation. This decomposition captures the topological information and can be used for further computation and refinement.

Keywords: Numerical algebraic geometry, cell decomposition, algebraic surface, algebraic curve, homotopy continuation, deflation

1 Introduction

Bertini_real seeks to automate the task of visualizing and computing on real algebraic curves and surfaces. From only a defining polynomial system, the program computes a cellular decomposition of the real portion of a one- or two-dimensional complex algebraic set. The output of Bertini_real is a set of text files, containing the set of computed vertices, the connections between them, and

any associated homotopies. Using the homotopies, the decomposition can be refined to the user's desire with a supplemental program simply titled `sampler`. An interactive visualization suite is provided in `MATLAB`.

`Bertini_real` works by leveraging the power of homotopy continuation [2, 3], numerical irreducible decomposition [8], regeneration [5], randomization [3], and isosingular deflation [6] to decompose the real parts of complex one- and two-dimensional components of algebraic varieties. It produces a cell decomposition, similar to the output of other decomposition methods, most notably the Cylindrical Algebraic Decomposition [1].

2 Functionality

`Bertini_real` is an MPI parallel-enabled compiled program called from the command line. The two necessary ingredients to run the software are: 1) a `Bertini` input file, and 2) a Numerical Irreducible Decomposition (NID) produced by `Bertini`. It further depends on `MATLAB` for symbolic calculations (*e.g.* deflation, symbolic derivatives and determinants), the Boost C++ support library, as well as GMP and MPFR (for multiple-precision numerics). Compilation requires a library-compiled version of `Bertini` [2].

The basic pattern for usage of `Bertini_real` is summarized below.

1. Create a NID, via `Bertini`. This gives a witness set for each irreducible component, as well as information on each component's degree, multiplicity, and deflation requirements.
2. Run `Bertini_real` on a single irreducible component. `Bertini_real` checks if the component is self-conjugate. If it is not, `Bertini_real` finds the intersection of the component with its conjugate and proceeds. The program further deflates the system [6] so that the component is reduced and properly deflated, so that we may track on it. It then finds a cell decomposition of the real points in the complex set.
3. Refine the decomposition. `Bertini_real` produces raw decompositions that are bare skeletons of the objects they describe. If the user wants to view a smoothed version, or use the decomposition for further calculations, they might want to refine using the program `sampler`, provided as part of the `Bertini_real` package.
4. Visualize. Visual interpretation of the data typically quickly reveals any problems which might have been encountered during computations. The suite of graphical software is provided through `MATLAB`.

3 Application

3.1 Curve

Consider a three-jointed revolute planar robot, with equal link lengths – and let the length be unity. If we fix a point in the workspace of the robot, we get

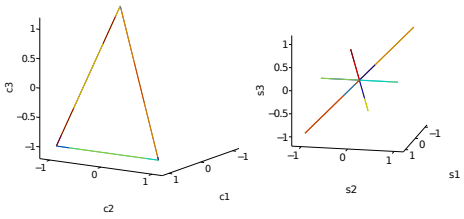

Fig. 1. Example of curve decomposition. A 3R planar robot of unit link length places its end effector at the point $(x, y) = (1, 0)$. Left: projection onto the cosines of the angles. Right: projection onto sines. These two plots are simpler than viewing the joint angles directly, due to the periodic nature of trigonometric functions.

a curve of solutions in terms of the joint angles such that the end effector is placed at the point. Equations are given below in (1), with $s_i = si = \sin\theta_i$, and $c_i = ci = \cos\theta_i$.

$$\begin{bmatrix} c_1 - s_3(c_1s_2 + c_2s_1) + c_1c_2 - s_1s_2 + c_3(c_1c_2 - s_1s_2) - 1 \\ s_1 + c_1s_2 + c_2s_1 + s_3(c_1c_2 - s_1s_2) + c_3(c_1s_2 + c_2s_1) \\ c_2^2 + s_1^2 - 1 \\ c_2^2 + s_2^2 - 1 \\ c_3^2 + s_3^2 - 1 \end{bmatrix} = 0 \qquad (1)$$

In Fig. 1, we present the three components of the solution curve when we grasp the point $(x, y) = (1, 0)$. On the left is a projection of the set onto the cosines, and the figure on the right are the sines.

3.2 Surface

Now consider a two-joint revolute planar robot with link lengths $\ell_1 = 1, \ell_2 = 0.5$, and let the target position for the end effector be variable and denoted (x, y), as in (2). The set of points in the plane the robot can reach is realizable using a surface decomposition. The workspace ought to be an annulus, and this is indeed the result of the decomposition when projected onto (x, y) as in Fig. 2.

$$\begin{bmatrix} c_1 - x + (c_1c_2)/2 - (s_1s_2)/2 \\ s_1 - y + (c_1s_2)/2 + (c_2s_1)/2 \\ c_1^2 + s_1^2 - 1 \\ c_2^2 + s_2^2 - 1 \end{bmatrix} = 0 \qquad (2)$$

4 Underlying theory

4.1 Curve

The implementation of curve decomposition in `Bertini_real` follows the algorithm laid out in [7], depicted in Fig. 3, and summarized informally below.

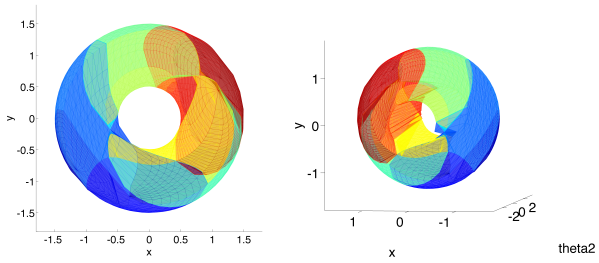

Fig. 2. Example of surface decomposition. A 2R planar robot with differing link lengths is allowed to move freely, and we decompose its workspace as a surface in terms of (x, y) and the sines and cosines of the joint variables. On the left, projection of the surface onto (x, y) gives an annulus as expected. At the right, the surface is tilted, revealing the two solutions, in terms of the arctangent of (s_2, c_2).

To begin, there is a little user set up, the foremost of which is to run `Bertini` with configuration setting `TrackType:1` to obtain a NID. Optionally, the user may write a file containing a (random) real projection and a sphere of interest. `Bertini_real` automatically tests for self-conjugacy. A non-self-conjugate component is intersected with its own conjugate to produce a finite set of isolated real points, which terminates the computation. Otherwise, `Bertini_real` carries out the following six steps.

1. Find critical points. These points will include singular points, and points such that the curve is tangent to the direction of projection, and they will satisfy the system:

$$f_{\text{crit}} = \begin{bmatrix} f(x) \\ \det \begin{pmatrix} Jf(x) \\ J\pi_1(x) \end{pmatrix} \end{bmatrix} = 0, \tag{3}$$

   where J indicates the Jacobian matrix of partial derivatives and $\pi_1 : \mathbb{C}^N \to \mathbb{C}$ is the random real projection being used for the decomposition. Let $c_1, \ldots, c_n$ be the real critical points, ordered so that $\pi_1(c_1) < \pi_1(c_2) < \cdots < \pi_1(c_n)$.
2. Intersect with sphere. To cut off unbounded arcs of the curve, or to focus the view to the user's region, we intersect with a sphere of center x_0 and radius r, and solve the system (4), inserting the real intersection points into the list of ordered critical points.

$$f_{\text{sphere}} = \begin{bmatrix} f(x) \\ ||x - x_0||_2^2 - r^2 \end{bmatrix}. \tag{4}$$

3. Slice. To find what will become the midpoints of the edges of the decomposition, slice the curve between its critical points, by tracking from the single witness linear $\mathcal{L}$ to each midpoint projection value, $p_{m_i} = (\pi_1(c_i) +$

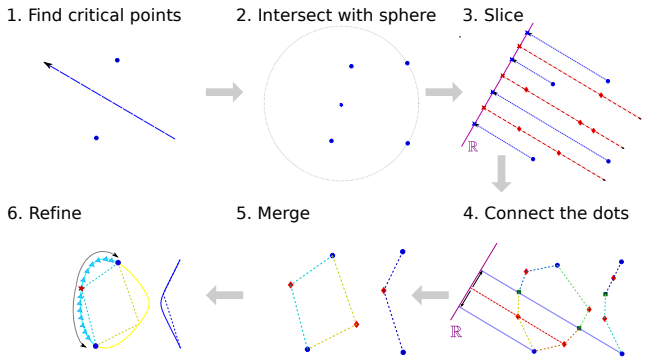

Fig. 3. The six major steps for a curve decomposition as implemented in `Bertini_real`. This illustration uses an elliptic curve, $x^3 - 2x + 1 - y^2 = 0$.

$\pi_1(c_{i+1}))/2$, as:

$$H_{\text{midslice}} = \begin{bmatrix} f(x) \\ t\mathcal{L}(x) + (1-t)(\pi_1(x) - p_{m_i}) \end{bmatrix}.$$

4. Connect the dots. Use the following homotopy to track midpoints first left and then right to the points on the curve above each critical point:

$$H_{\text{track}} = \begin{bmatrix} f(x) \\ \pi(x) - (tp_{m_i} + (1-t)p) \end{bmatrix},$$

where p is taken first as $p = \pi_1(c_i)$ and then as $p = \pi_1(c_{i+1})$.
5. Merge. Optionally, we can remove superfluous intersections which lie in the same projection fiber as critical points. These points arise when the curve has non-critical branches above a critical point, and they can be removed to produce a simpler decomposition.
6. Refine. Optionally, the user can refine the decomposition to their specification. By using the same homotopy as in Step 4, we can move the generic point in the center of each edge to any projection value p, $\pi_1(c_i) < p < \pi_1(c_{i+1})$. Two methods are available in `Bertini_real`: 1) a fixed-number method, where the user specifies how many points they want per edge; and 2) an adaptive method, where the user specifies a distance tolerance and a limit on the of number of refinement iterations.

4.2 Surface

The implementation of surface decomposition in `Bertini_real` follows the algorithm laid out in [4], depicted in Fig. 4, and summarized informally below.

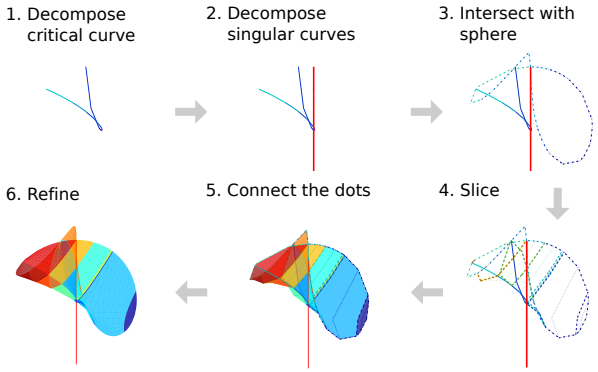

Fig. 4. The six major steps for a surface decomposition as implemented in `Bertini_real`. This example uses the Whitney Umbrella, $x^2 - y^2 z = 0$, a degree 3 surface in three variables, which is unbounded and contains a curve of singularity (around part of which, the surface is one-real dimensional.

Similarly to a curve decomposition, there is a small amount of user set up. Of course, one must obtain a NID, and the user may choose a projection and sphere. Self-conjugacy testing, and deflation are performed automatically. The six steps below are for self-conjugate components only. Any non-self-conjugate component is intersected with its conjugate component, producing at most a curve, which is then treated as in the curve case above. The decomposition of a surface is found with respect to two random real projections, $\pi_1, \pi_2 : \mathbb{C}^N \to \mathbb{C}$, as follows.

1. Decompose the critical curve. The critical curve is analogous to the outline of an object when viewed in an image plane and is also the set where the tangent is parallel to the *two* directions of projection, π_1, π_2. The curve is defined by the system:

$$f_{\text{critcurve}} = \left[\det \begin{pmatrix} f \\ Jf(x) \\ J\pi_1(x) \\ J\pi_2(x) \end{pmatrix} \right] = 0. \tag{5}$$

   Witness sets for the components of the critical curve are obtained via regeneration from the witness set, using a left-nullspace approach, and these are passed the curve method for decomposition with respect to π_1.
2. Decompose singular curves. As a matter of course from computing the witness set for the critical curve, we also obtain witness points for singular curves, since every singular curve will also satisfy (5). We use isosingular deflation [6] to deflate the input system these witness points, thereby producing

full witness sets. These are then decomposed with respect to π_1 exactly as for any other component of the critical curve.

3. Intersect with sphere. The intersection of the surface with a sphere of radius r and center x_0 will result in a curve, defined by (4). The intersection curve is treated as part of the critical curve, so it too is decomposed with respect to π_1.
4. Slice. We perform a curve decomposition at each of two sets of π_1 projection values – *at* each critical π_1-value, and *halfway between* each pair, coming from the critical points of the critical curve, singular curves, and the sphere curve. Call these *critical slices* and *mid-slices*, respectively. Each of these slices has a constant π_1 value and is decomposed with respect to π_2.
5. Connect the dots. The midpoints of each edge of each mid-slice become the center point for a face of the decomposition. The decompositions of the mid-slices reveal how the midpoint is connected to the top and bottom edges of its face, each coming from the critical curve, the sphere curve, or a singular curve. The description of the face is completed by finding which edges in the adjacent left and right critical slices connect to the midpoint. This is determined using a homotopy that keeps the midpoint from crossing its top and bottom edges as it is moved to the left and right critical projection values: see [4].
6. Refine. The decomposition to this point is coarse, in that it provides a coarse triangulation of the surface. A refinement method is provided in the separate executable `sampler`, which refines each edge and face in the decomposition, to contain a number of points of the user's choice. Adaptive and eventually optimal sampling for surfaces is a matter of ongoing development.

5 Technical contribution

5.1 Advances

`Bertini_real` allows a non-expert access to the algorithms of [7, 4] for decomposing the real points of complex algebraic curves and surfaces, whereas the previous prototype codes required expertise and worked only on sets of low degree. Importantly, `Bertini_real` is the first implementation that removes the restriction to almost-smooth surfaces that was needed in [4] — non-smooth surfaces can now be treated in any number of variables. The largest curve we have decomposed so far is a 3-3 Burmester curve [9] in 14 variables of degree 630.

5.2 Challenges

The main algorithms as implemented in `Bertini_real` are all for affine varieties. One can decompose any projective variety one wants, by considering patch equations and the transformation into an affine space. However, the `Bertini` tracker loops used by `Bertini_real` expect there to be a single homogenizing variable for a single non-homogeneous variable group. Furthermore, `Bertini` as written

was not intended to be called as a library as we do with `Bertini_real`, so linking into the loops required a great deal of finesse. This experience is helping the setting up of specifications for the next version of `Bertini`.

While curves are comparatively easy to decompose, `Bertini_real`'s surface decomposer is currently capable of dealing with only moderately sized systems — surfaces involving no randomization and six variables are generally currently tractable. However, we have encountered difficulty decomposing a particular Burmester surface, involving eight polynomials in ten variables. While we can readily obtain the witness points for the critical curve, computing the critical points of the critical curve remains a barrier for this problem. The code uses a determinantal formulation of the criticality condition, wherein we compute a symbolic determinant involving a Jacobian matrix. `MATLAB` struggles with this, eventually spitting out a system over 25 MB in size. Worse, `Bertini` must then parse this input file to create procedures for evaluating the function and its Jacobian, which overwhelms the available computing resource.

The major obstacle to running large problems through the surface decomposer is therefore the elimination of the determinant. Alternate methods that avoid the determinant are the subject of further research.

Acknowledgments All authors were partially supported by AFOSR grant FA8650-13-1-7317. DJB was partially supported by NSF grant DMS-1025564. DAB and JDH were additionally supported by DARPA YFA.

References

1. 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.
2. D.J. Bates, J.D. Hauenstein, A.J. Sommese, and C.W. Wampler. "`Bertini`: Software for Numerical Algebraic Geometry." Available at `bertini.nd.edu`.
3. D.J. Bates, J.D. Hauenstein, A.J. Sommese, and C.W. Wampler. *Numerically solving polynomial systems with Bertini*, volume 25. SIAM, 2013.
4. 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.
5. 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.
6. J.D. Hauenstein and C.W. Wampler. Isosingular sets and deflation. *Foundations of Computational Mathematics*, 13(3):371–403, 2013.
7. 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.
8. 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.
9. Y. Tong, D. H. Myszka, and A. P. Murray. "Four-bar linkage synthesis for a combination of motion and path-point generation." *Proc. ASME IDETC/CIE 2013*, Portland, OR, Aug. 4–7, 2013.