Bertini_real

software for real algebraic sets

Manual written by

Dani Brake
Pierce Cunneen
Elizabeth Sudkamp
Christopher Lembo

Contents

1 Introduction

Welcome to Bertini_real, software for real algebraic geometry. This manual is intended to help the user operate this piece of numerical software, to obtain useful and high-quality results from decomposing real algebraic curves and surfaces.

Bertini_real is compiled software, links against a parallel version of Bertini 1 compiled as a library (`libbertini-parallel`, and requires Matlab and the Symbolic Computation toolbox. It also requires several other libraries, including a few from Boost, and an installation of MPI. All libraries should be compiled using the same compilers and dependent libraries.

1.1 About this manual

The purpose of this manual is to provide a robust, orderly, and easy to understand instructions on how to use Bertini_real. This manual has three roles: it first serves as a description of what Bertini_real is, followed by instructions on Bertini_real's installation on Mac, Linux, and PC operating systems, and finally as a general reference manual for Bertini_real.

This manual is here to help guide a user through the installation process, as well as act as the user's manual for Bertini_real. If there is a section that might not be entirely clear, or is confusing to a reader, please contact us (see below for contact information), and we will try to resolve the problem. Such feedback is welcome!

1.2 Bertini_real product description

Bertini_real is an implementation of several numerical algorithms [5, 2], to decompose the real part of a complex curve or surface in any (tractible) number of variables. Some of the important features of Bertini_real include :

- It 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.
- It seeks to automate the visualization and computation of algebraic curves and surface.

1.3 Where Bertini_real can be found

The tarball for Bertini_real can be downloaded at Bertini_real.com. The visualization codes for MATLAB, they can be found at GitHub.

1.4 Who is developing Bertini_real?

Bertini_real is under ongoing development by the development team, which consists of Daniel Brake (University of Notre Dame), Daniel Bates (Colorado State University), Jonathan Hauenstein (University of Notre Dame), Wenrui Hao (Mathematical Biosciences Institutes), Andrew Sommese (University of Notre Dame), Charles Wampler (General Motors. R&D), and Pierce Cunneen (University of Notre Dame)

This manual was written by Daniel Brake, Pierce Cunneen, and Elizabeth Sudkamp.

1.5 Contact

Daniel Brake: danielthebrake@gmail.com – Main implementer
Daniel Bates: bates@math.colostate.edu – Advisory
Jonathan Hauenstein: hauenstein.edu – Advisory
Wenrui Hao: hao.50@mbi.osu.edu – Advisory

Andrew Sommese: sommese@nd.edu – Advisory
Charles Wampler: charles.w.wampler@gm.com – Advisory
Pierce Cuneen: pcuneen@nd.edu – Users manual
Elizabeth Sudkamp: esudkamp@nd.edu – Users manual

1.6 Acknowledgements

The development of Bertini_real has been supported generously by a number of sources, including

- the Vincent J. and Annamarie Duncan Professor of Mathematics, at the University of Notre Dame,
- the University of Notre Dame,
- the National Science Foundation grants DMS-1025564 , DMS-1115668, and DMS-1262428,
- the Air Force Office of Scientific Research grant FA8650-13-1-7317,
- Mathematical Biosciences Institute,
- the Sloan Research Fellowship,
- the Army Young Investigators Project, and
- the Defense Advanced Research Projects Agency Young Faculty Award.

Finally, funding for shared facilities used in this research was provided by the Division of Computer and Network Systems: an NSF grant under award number CNS-0923386.

Disclaimer

2 Quick Summary

Here's a super brief description of how to get and use this software:

Bertini_real can be downloaded from `http://bertinireal.com/download.html`, or cloned from its Github repo. Use of Bertini_real depends on Bertini, which itself has several important dependencies (see section A) Once installed, you can run Bertini_real on an input file from the command line. After navigating to the working directory of the input file, the flow of Bertini_real is as follows:

1. Run Bertini on an input file using the `tracktype:1` setting. This is done by typing in the command line: `bertini` with an input file named `input`. Bertini will produce a Numerical Irreducible Decomposition that will be used by Bertini_real.

2. Run Bertini_real on the same input file. Similarly, just type `bertini_real` in the command line. Bertini_real will provide a cellular decomposition of the real portion of a one- or two-dimensional complex algebraic set.

3. Visualize the results of Bertini_real in Matlab. Enter Matlab and call `gather_br_samples`, which parses the output results of into a .mat file, and then call `bertini_real_plotter`, which will plot the curve or surface in Matlab. Please note that the Matlab executable must be on the path to the input file for Bertini_real to run).

3 Input Files

The instructions provided go through how to create input files, run these files through Bertini and Bertini_real, and view a graphical representation of the results using MATLAB. Most of this information about Bertini, its input files, and syntax will be taken or paraphrased from the slightly-out-of-date Bertini User's Manual, which can be read here.

The `input` file has two parts, grouped as follows (where the % symbol is the comment character in the `input` file, as usual):

```
CONFIG
% Lists of configuration settings (optional)
tracktype:1; % needed in order to run Bertini_real
END;
INPUT
% Symbol declarations
% Optional assignments (parameters, constants, etc.)
% Function definitions
END;
```

File 1: Adapted from [1]

The upper portion of the file consists of a list of configuration settings. Any configuration that is not listed in the `input` file will be set to its default value. A table of all configuration settings that may be changed, along with their default settings and acceptable ranges, may be found in the Appendix.

The syntax for the configuration lines is straightforward. It consists of the name of the setting (in all caps), followed by a colon, a space, the setting, and a semicolon. For example, to change the tracking type to 1 (the default is 0), simply include the following line in the `CONFIG` portion of the `input` file:

```
TRACKTYPE: 1;
```

File 2: Adapted from [1]

The lower portion of the `input` file begins with a list of symbol declarations (for the variables, functions, constants, and so on). All such declarations have the same format:

```
KEYWORD a1, a2, a3;
```

File 3: Adapted from [1]

where `KEYWORD` depends upon the type of declaration. All symbols used in the `input` file must be declared, with the exception of subfunctions. Here are details regarding each type of symbol that may be used in the input file:

- FUNCTIONS: Regardless of the type of run, all functions must be named, and the names must be declared using the keyword `function`. Also, the functions must be defined in the same order that they were declared.

- VARIABLES In all cases except user-defined homotopies, the variables are listed by group with one group per line, with each line beginning with either the keyword `variable_group` (for complex variable groups against which the polynomials have not been homogenized) or the keyword `hom_variable_group` (for variable groups against which the polynomials have been homogenized). Note that the user must choose one type of variable group for the entire input file, i.e., mixing of variable groups is not allowed in this release of Bertini. Also, only one variable group may be used for a positive-dimensional run. For example, if there are two nonhomogenized variable groups, the appropriate syntax would be

```
variable_group z1, z2;
variable_group z3;
```

File 4: Adapted from [1]

In the case of user-defined homotopies, the keyword is `variable`, and all variables should be defined in the same line.

- PATHVARIABLES: The pathvariable, often denoted by the letter "t", is the independent variable that is controlled during homotopy continuation. In Bertini, the homotopy always moves from the start system at $t = 1$ to the target system at $t = 0$. A pathvariable must be declared in the `input` file **ONLY** if the user is specifying the entire homotopy (i.e., `USERHOMOTOPY` is set to 1). In that case, it is also necessary to declare at least one parameter, as described in the next item. The keyword for pathvariables is `pathvariable`.

- PARAMETERS: Homotopy continuation relies on the ability to cast a given polynomial system as a member of a parameterized family of polynomial systems. Such parameterized families (especially those which occur naturally) constitute one of the powerful advantages numerical methods in algebraic geometry have over symbolic methods. Sometimes there is only one parameter involved, but sometimes there are several. Please note, though, that user-defined parameters should be used only in the case of user-defined homotopies. Regardless of the number of parameters, each parameter depends directly upon the pathvariable. As a result, the user must both declare each parameter and assign to it an expression depending only upon the pathvariable to it. Here is an example:

```
...
parameter p1, p2;
...
p1 = t^2;
p2 = t^3;
...
```

File 5: Adapted from [1]

For technical reasons, in the case of a user-provided homotopy, Bertini always assumes that there is at least one parameter (even if there is no apparent need for one). In the case that the user wishes to build a homotopy depending only upon the pathvariable, it is necessary to declare a parameter, set it to the pathvariable in the assignments section, and then to use only that parameter (and **NOT** the pathvariable) in the functions. Here is an example:

```
...
pathvariable t;
parameter s;
...
s=t;
...
```

File 6: Adapted from [1]

No parameters should appear in the `input` file unless the homotopy is defined by the user, and the pathvariable should never appear explicitly in any homotopy function.

- CONSTANTS: Bertini will accept numbers in either standard notation (e.g., 3.14159 or 0.0023) or scientific notation (e.g., 3.14159e1 or 2.3e-3). No decimal point is needed in the case of an integer. To define complex numbers, simply use the reserved symbol I for $\sqrt{-1}$, e.g., `1.35 + 0.98*I`. Please note that the multiplication symbol * is always necessary, i.e. concatenation does not mean anything to Bertini. Since it is sometimes useful to have constants gathered in one location (rather than scattered throughout the functions), Bertini has a `constant` type. If a constant type is to be used, it must be both declared and assigned to. Here is an example:

```
...
g1 = 1.25;
g2 = 0.75 - 1.13*I;
...
```

File 7: Adapted from [1]

Bertini will read in all provided digits and will make use of as many as possible in computations, depending on the working precision level. If the working precision level exceeds the number of digits provided for a particular number, all further digits are assumed to be 0 (i.e., the input is always assumed to be exact). This seems to be the natural, accepted implementation choice, but it could cause difficulty if the user truncates coefficients without realizing the impact of this action on the corresponding algebraic set.

- SUBFUNCTIONS: Redundant subexpressions are common in polynomial systems coming from applications. For example, the subexpression `x^ 2 + 1.0` may appear in each of ten polynomials. One of Bertini's advantages is that it allows for the use of subfunctions. To use a subfunction, simply choose a symbol, assign the expression to the symbol, and then use it in the functions. There is no need to declare subfunctions (and no way to do so anyway).

```
...
V = x/2 + 1.0;
...
f1 = 2*V^2 + 4.0;
...
```

File 8: Adapted from [1]

- SIN, COS, PI, AND EXP: Starting with Bertini v1.2, the sine function sin, cosine function cos and exponential function exp are built into Bertini. Additionally, Bertini uses `Pi` for the constant π. To avoid confusion with scientific notation, the constant e is not specifically built in Bertini, but the user can define their own constant and set it equal to `exp(1)`, as shown below.

```
...
constant EN; % Euler's␣number␣e
EN␣=␣exp(1);
...
```

File 9: Adapted from [1]

It is important to note that Bertini will return an error if the argument of `sin`, `cos`, or `exp` depends upon a variable when trying to solve a polynomial system. There is no such restriction for user-provided homotopies.

3.1 On Bertini input syntax

Common complaints about Bertini are that (a) the parser that reads in the input is very picky and (b) the error messages are often to general. The development team agrees and will continue to work on this (especially during an upcoming complete rewrite). In the meantime, here is a list of syntax rules that are commonly broken, resulting in syntax errors:

- All lines (except `CONFIG` and `INPUT`, if used) must end with a semicolon.
- Bertini is case-sensitive.
- The symbol for $\sqrt{-1}$ is `I`, not `i`. If you prefer to use `i`, you may define `i` as a subfunction by including the statement `i = I;`.
- In scientific notation, the base is represented by `e` or `E`, e.g., 2.2e-4.
- For multiplication, * is necessary (concatenation is not enough).
- Exponentiation is designated by ˆ.
- All symbols except subfunctions must be declared prior to use. You cannot combine declaration and definition, sadly.
- No symbol can be declared twice. This error often occurs when copying and pasting in the creation of the input file.
- A pathvariable and at least one parameter are needed for user-defined homotopies. Please refer to the previous section for details.
- White space (tabs, spaces, and new lines) is ignored.

4 Numerical Irreducible Decomposition

Bertini_real takes as input a Numerical Irredicible Decomposition (NID) of the complex algebraic variety for your problem. The NID is computed by Bertini, `tracktype: 1`, and is stored in a file called `witness_data`.

Once your input file describing the system you want to solve is created, you need to run Bertini. Navigate in the command line to the directory of the `input` file and type `bertini` or `bertini your_input_file_name`. Bertini assumes the input file is named `input` unless told otherwise, by passing the name as the first argument. No, you do not currently use flags to specify input file name.

Cygwin users: A user may also use `bertini-serial.exe` (or `bertini_parallel.exe`). This will run Bertini, creating the Numerical Irreducible Decomposition needed for Bertini_real. You may need to type in the entire pathway to where Bertini is located, if it's not in the same folder, so the command line read
`/cygdrive/path/to/BertiniSource_v1.5/bertini-serial.exe input`

If the NID run is successful, you should see a summary of the decomposition print to the screen. It should look something like this:

```
************* Witness Set Decomposition *************

| dimension | components | classified | unclassified
-----------------------------------------------------
|    2      |    1       |    7       |   0
-----------------------------------------------------

************** Decomposition by Degree **************

Dimension 2: 1 classified component
-----------------------------------------------------
   degree 7: 1 component

*****************************************************
```

File 10: Example NID output, tracktype 1 in Bertini 1

If there are path failures or unclassified points, change Bertini settings, and re-run the problem. Consult the Bertini book or user's manual for more information about available settings, and their impact on computing the NID.

4.1 Necessary Bertini output files for Bertini_real

The main output file of interest from Bertini is called `witness_data`, a file suited for automated reading by a program. and terribly formatted for humans. It contains all of the information needed to describe the witness sets for the irreducible components of your variety. In particular, it has the information used for regeneration used in Bertini_real, as well as component sampling and membership testing.

Do not rename `witness_data`.

5 Running Bertini_real

Bertini_real is called from the command line. This is done simply by calling `bertini_real` (or `bertini_real.exe` for Cygwin users) from the command line. If the input file is called anything other than `input`, than the `-input` or `-i` option followed by the filename must be used.

It is important to note that for Bertini_real to run, the MATLAB executable must be on the path.

Bertini_real uses the tracker options for Bertini, which are set at the top of the input file, in the `CONFIG` section.

We suggest the following configuration options in the input file for Bertini_real:

- `sharpendigits` ≈ 30

  helps keep regeneration start points on target, and helps identify points which are supposed to be the same point.

Other options can improve performance and tighten up the produced decomposition.

5.1 Files Needed for Input

In order to sucessfully run Bertini_real, the program needs to be able to access the original `input` file that was used in Bertini, as well as the `witness_data` file generated by Bertini.

5.2 Command prompt, options

There are a number of inline commands that can be used while running Bertini_real. Below is a table that describes these options:

Table 1: Bertini_real command line options

Option	**Alter**	**Command Line**	**Description**
`-component`	integer index of the component	`bertini_real -component 1`	Decomposes only one component of the entire figure
`-debug`	n/a	`bertini_real -debug`	If used, program will pause for 30 seconds before running for debugging purposes
`-dim` or `-d`	target dimension of solution	`bertini_real -d 2`	Sets a target dimension to be used for the solution

Continued on next page

Table 1: *Continued from previous page*

Option	**Alter**	**Command Line**	**Description**
`-gammatrick` or `-g`	`1` (if you'd like Bertini_real to use the gamma trick) or `0` (if not)	`bertini_real -g 1`	Indicator for whether Bertini_real should use the gamma trick in a particular solver
`-help` or `-h`	n/a	`bertini_real -h`	Displays a help message containing the version of Bertini_real, where Bertini_real can be found online, support information, and finally the command line options.
`-input` or `-i`	filename	`bertini_real -i myfile`	Used if input file is named something other than 'input'
`-mode` or `-m`	`bertini_real` (default) or `crit`	`bertini_real -m crit`	Sets the mode of Bertini_real to be used
`-nostifle` or `-ns`	n/a	`bertini_real -ns`	If used, screen output will not be stifled
`-nomerge` or `-nm`	n/a	`bertini_real -nm`	Indicates that Bertini_real should not merge ends
`-output` or `-out` or `-o`	name of the output directory	`bertini_real -out bertinir_results`	Places the output files in a different directory
`-projection` or `-pi` or `-p`	desired filename	`bertini_real -p myprojection`	Indicator for whether to read the projection from a file, rather than randomly choose it
`-quick` or `-q`	n/a	`bertini_real -q`	Solves problem quickly, but not as robust

Continued on next page

Table 1: *Continued from previous page*

Option	**Alter**	**Command Line**	**Description**
`-veryquick` or `-vq`	n/a	`bertini_real -vq`	Solves problem very quickly, but not as robust
`-sphere` or `-s`	the name of the file for Bertini_real to read	`bertini_real -sphere mysphere`	Sets indicator that Bertini_real should use sphere created by user rather than just compute sphere
`-verb`	the level of the verbosity	`bertini_real -verb 2`	Shows or hides output text
`-version` or `-v`	n/a	`bertini_real -version`	Displays the version of Bertini_real running on your computer

Br15

5.3 Parallelism

Bertini_real is parallel-enabled, using MPI (but not OpenMP or threads). To use multiple processors, call it as you would any other MPI program: `mpiexec [options] bertini_real`.

5.4 Projections and spheres of interest

Here we describe the `pi` file in Section 5.4.1, and a `sphere` of interest in Section 5.4.2.

5.4.1 The user-defined projection, `pi`

The `pi` file, defining a specific projection to use for decomposing your curve or surface, has a simple format. You indicate the number of variables, and then give the projection. No punctuation or delimiters necessary.

Using a particular projection, contained in a file of arbitrary name, is indicated to Bertini_real by passing the `-pi` flag. For example, `bertini_real -pi my_projection`.

By default, Bertini_real uses a randomly generated projection to decompose the object. This is so that the object is in *general position*, which is required for set-of-measure-zero guarantees that all elements of the critical space lie in the distinct fibers of the projection.

For decomposing surfaces, if you feel the need to supply your own projection, please consider using two projections π_1 and π_2 such that $\pi_1 \cdot \pi_2 = 0$. That is, orthogonal projections tend to produce cleaner decompositions.

```
num_coords    <--- the number of variables in the problem

pi_1_1  <--- curves and surfaces need at least one projection
pi_1_2
...
pi_1_N

pi_2_1   <---  only surfaces need a second projection
pi_2_2
...
pi_2_N   <--- as many entries in each projection as there are
    coordinates
```

File 11: A file describing a user-defined projection used to decompose a real object in 4 dimensions. The first number indicates the number of coordinates, which must match the number in the object's ambient space. Then, the values for the projection. Try to use orthogonal projections for surfaces.

5.4.2 The sphere of interest, `sphere`

While Bertini_real will happily compute a bounding sphere for you, containing all the interesting parts of your object, it may be very large, or kind of wonky in the case of some projections. Hence, we allow the user to specify their sphere of interest by way of plain text file.

The `sphere` file allows the user to bound the space in which to decompose their object. If there is a region of space you are interested in, you can compute your object inside a sphere of interest.

To inform Bertini_real that you are using your own sphere rather than the computed one, use the `-sphere` flag. For example, `bertini_real -sphere my_sphere_file`. This generally will not speed up computation at all, since there's no way to know prior to point computation whether the endpoint will be in or out of the sphere. All it will change is the bounded region.

```
radius

x_1_center
x_2_center
...
x_N_center
```

File 12: A file describing a sphere of interest to Bertini_real. The radius appears first, followed by the coordinates of the center of the sphere. Real coordinates only, omit the imaginary part.

6 Running Sampler

If you are happy with the results of the Bertini_real decomposition, you may wish to refine the triangulation of the surface or curve. This can be acheived using the `sampler` program after calling `bertini_real`. Call `sampler` on the command line, e.g. `sampler -fixed 10` to sample each cell to have approximately 10^d samples on it, where d is the dimension of the component.

7 Visualization

7.1 In Matlab

After running Bertini_real, the output results can be visualized in Matlab. This section assumes that the Matlab codes for Bertini_real are already on the Matlab path.

1. First, open Matlab, move to the folder in which you decomposed your object, and call `gather_br_samples`. This parses the output from Bertini_real into a .mat file.

2. Then, call `bertini_real_plotter`, which creates a handle class object and facilitates selection of parts of the decomposition to view. There are many options, all of which are documents and displayed via `help bertini_real_plotter` in Matlab.

3. To run bertini_real_plotter with a specific option, type in Matlab `bertini_real_plotter('option', 'option_argument')`, where the option_argument will vary depending on the option you decide to alter. The options are listed below.

7.2 Visualization options

Table 2: MATLAB Visualization Options

Option	**Default**	**Alter**	**Command Line**	**Description**
`'autosave'`	`'on'`	`'false'`, `'0'`	`bertini_real_plotter ('autosave', 'false')` off	Users can automatically save a figure to the working directory or not.
`'colormap'`	`'jet'`	full list here	`bertini_real_plotter ('colormap', @summer)` summer colormap	Users can change the colormap by changing the handle.
`'curve'` or `'curves'`	`'true'`	`'n'`, `'no'`, `'none'`, `'false'`, `'0'`	`bertini_real_plotter ('curve', 'false')` disables the curves option	`bertini_real_plotter`- by default lets the user display the figure's raw curves.
`'faces'`	`'true'`	`'n'`, `'no'`, `'none'`, `'false'`, `'0'`	`bertini_real_plotter ('faces', 'none')` makes only the option to display the raw curves will be given.	By default, the figure created in MATLAB will show both the raw curves and faces.

Continued on next page

Table 2: *Continued from previous page*

Option	**Default**	**Alter**	**Command Line**	**Description**
'filename' or 'file'			bertini_real_plotter ('filename', 'Example_File_Name.mat')	bertini_real_plotter- first searches files named BRinfo∗.mat; if more than one, uses most recent
'labels'	'on'	'n','no', 'none', 'false', '0'	bertini_real_plotter ('labels', 'none') off	bertini_real_plotter- by default lets user apply labels to the figure.
'linestyle'	'-' (solid line)	line options listed here	bertini_real_plotter ('linestyle', ':')	Used to change the line style of lines in the MATLAB figure.
'monocolor' or 'mono'	'off'	RGB triples listed here	bertini_real_plotter ('mono', 'r') creates a red figure	Used to create a mono-color figure.
'proj'				Unsure of what proj does
'vertices' or 'vert'	'on'	'n', 'no', 'none', 'false', '0'	bertini_real_plotter ('vertices', 0) off	MATLAB can allow the user to place vertex markers and labels on the figure.

Br15

7.3 3D Printing

The following description on how to 3D print the surfaces that were decomposed from Bertini_real is taken from *Bertini_real: Numerical Decomposition of Real Algebraic Curves and Surfaces.*[3]

We strive to make three-dimensional printing of surfaces decomposed using Bertini real almost trivial. Since even the unrefined surface decomposition is a triangulation, 3D printing is as simple as:

- Convert data from plaintext
- Write to stereolithography (stereolithography (STL)) file
- If not compact, solidify the surface
- Process in model repair service
- Graphically ensure quality
- 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.[3]

8 Troubleshooting

8.1 Installation

- Compilation fails, with an error due to send calls for MPI.

  You are probably using OpenMPI, and their implementation does not use `const`-correctness. Dani needs to modify the code in `bertini_extensions` to cast away constness for these send calls. Send him an email to poke him. Or, use a different implementation of MPI. MPICH2, versions 3.04 and up are known to compile using the code as written.

8.2 Bertini

8.3 Bertini_real

8.3.1 `had a critical failure`

Missing critical points, or curve slicing problems Sometimes when decomposing an object, Bertini_real will display something like

```
had a critical failure moving left was deficient 2 points trying to recover the failure...
tracktolBEFOREeg: 1e-07 tracktolDURINGeg: 1e-08
```

This is generally due to a missing critical point. Bertini_real uses linear product regeneration to compute critical points, and a missing critical point will cause errors as above.

The solution to this problem is to get Bertini to not miss any critical points. This means understanding the path tracker, and what settings influence tracking success. I generally find several settings can positively influence computation of critical points:

- `securitymaxnorm`, `securitylevel`

  This is the norm of path truncation during the endgame, during path tracking in Bertini. If two successive approximations of points on a path exceed this norm, the path will be truncated, unless `securitylevel` is set to `1`. Setting the security level to 1 will make tracking take longer, as all paths which end at ∞ are tracked all the way to the end. So level 0 spares computation. However, during computation of critical points, synthetic variables representing nullspaces of Jacobians are used, and these can have large norms, resulting in truncation of paths which we need to succeed to fully decompose the object.

  Hence, we recommend setting securitymaxnorm to something large but not crazy large. This is naturally problem dependent. Somewhere around 1e14 has proven useful in our experiments. YMMV.

  To prevent any paths from being truncated, use `securitylevel 1`.

- `condnumthreshold`

  The post-processor in Bertini classifies endpoints of paths as singular on several criteria, including multiplicity, and on condition number of the Jacobian. To prevent classification of endpoints as singular, raise this threshold. Sometimes large values are needed, upwards of perhaps 1e30 or beyond.

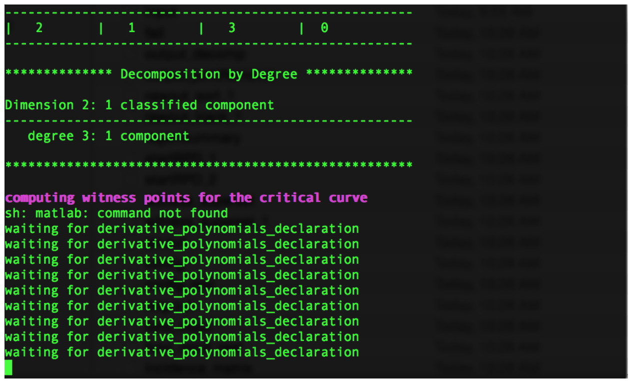
- `sharpendigits`

  After tracking, for nonsingular endpoints Newton's method can be run to increase the accuracy of approximations. This setting sets the number of digits you wish for. You are not guaranteed this number, because numerical conditioning may prevent sharpening from completing.

8.4 Matlab

8.4.1 `sh: matlab: command not found`

If you get the message `sh: matlab: command not found` from running Bertini_real, then Matlab is not on your shell path. Bertini_real currently requires Matlab to run properly, and thus failure to include the Matlab executable on the path will cause bertini_real to fail. Below is an example of the terminal output displayed when Bertini_real is unable to locate the Matlab executable:



```
-----------------------------------------------------
|   2       |   1       |   3       |   0
-----------------------------------------------------

************** Decomposition by Degree **************

Dimension 2: 1 classified component
-----------------------------------------------------
   degree 3: 1 component

*****************************************************

computing witness points for the critical curve
sh: matlab: command not found
waiting for derivative_polynomials_declaration
waiting for derivative_polynomials_declaration
waiting for derivative_polynomials_declaration
waiting for derivative_polynomials_declaration
waiting for derivative_polynomials_declaration
waiting for derivative_polynomials_declaration
waiting for derivative_polynomials_declaration
waiting for derivative_polynomials_declaration
waiting for derivative_polynomials_declaration
```

Permanently fixing this error involves editing your profile, e.g. `bash_profile`. From your home directory, open `.bash_profile` and add the following line: `export PATH=/PATH/TO/MATLAB.app/bin:$PATH` where `PATH/TO/MATLAB.app` points to the location of the Matlab executable. If you do not have, type `touch .bash_profile`, which will create `.bash_profile`, and then add `export PATH=/PATH/TO/MATLAB.app/bin:$PATH`
This should result in the Matlab executable being added to the path whenever opening terminal.

8.4.2 Calling Bertini_real from within Matlab

Note that you cannot run Bertini_real from within Matlab, even with the use of `!`, because Bertini_real currently calls Matlab using a `system()` call.

8.4.3 Removing dependency on Matlab's symbolic toolbox

Removal of Matlab as a dependency is ongoing work. Please consider helping me move to Python or another open source language for the symbolic computations, required for deflation of singular curves, and the writing of the input file for critical curves of surfaces.

9 Examples

9.1 Curves

9.2 Surfaces

References

[1] Dan Bates and Jon Hauenstein. *Bertini Manual*, 2013.

[2] 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.

[3] Daniel Brake, Daniel Bates, Wenrui Hao, Jonathan Hauenstein, Andrew Sommese, and Charles Wampler. Bertini real: Numerical decomposition ofreal algebraic curves and surfaces. February 2015.

[4] Daniel Brake and Pierce Cunneen. *Bertini_real Manual*, 2015.

[5] 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.

[6] MartyMacGyver. How to install a newer version of gcc, July 2011.

A Install

This section of the manual focuses on how to install the necessary dependencies and programs needed to run Bertini_real on a user's computer. The instructions provided describe the process for Linux, Mac, and Windows operating systems. If you want to try to port Bertini_real to another operating system, please contact Daniel Brake.

When installing Bertini_real, there are a number of steps required in order to successfully install and run the program. They are:

1. Installing the dependencies
2. Installing Bertini
3. Installing Bertini_real

A.1 Dependencies

Before installing Bertini and Bertini_real, there are a number of packages that need to be installed. The method used to install these dependencies changes depending on the operating system, so please be sure to read the section that describes your particular system. The dependencies are:

- a C++ compiler capable of the C++ 11 standard
- an Message Passing Interface (MPI) (such as MPICH2)
- Boost >= 1.53
- Multiple Precision Floating-Point Reliable (MPFR)
- GNU Multiple Precision Arithmetic Library (GMP)

[4, p. 4]

MATLAB

The program MATLAB also needs to be installed on your computer as well. Instructions on how to install the program are not provided here. However, if you are associated with a university, or a research facility, they probably have download instructions on their technology support website (e.g. Notre Dame's OIT website).

In Windows

After installing MATLAB, please be sure to add `C:\User\username\...\matlab.exe` to your PATH variable.

In Unix GNU Linux

If you do not already have MATLAB on the PATH, type `touch .bash_profile`, which will create `.bash_profile`, and then add `export PATH=/PATH/TO/MATLAB.app/bin:$PATH`. This should result in the MATLAB executable being added to the path whenever opening terminal.

If you are a Windows user, the instructions on had to add MATLAB to the PATH are included in the Windows Instructions section

A.2 Instructions for GNU/Linux

Use the package manager provided, such as apt-get, to install the dependencies into your preferred directory.

A.3 Instructions for OSX

It is recommended that Mac users install the program Homebrew to use to install these packages. Once that has been done, installing the previously listed dependencies becomes simple. In terminal, type `brew search ____` to list packages related to ____, where ____ is your search (for example, GMP, Boost, or MPICH2). To download via Homebrew, type in terminal `brew install ____`

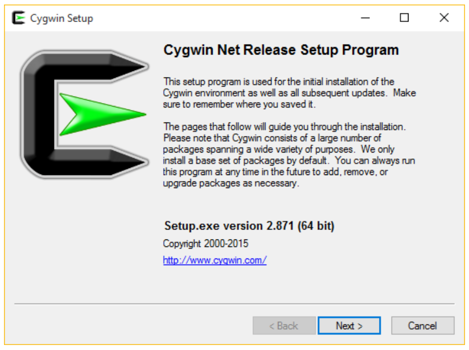
A.4 Instructions for Windows

Unlike Linux or Mac computers, Windows users have additional pre-requisites that they need to install in order to use Bertini and Bertini_real- they need to first install the program Cygwin. Alternatively, with Windows 10 and Bash support upcoming, consider using Chocolatey or bash itself.

A.4.1 Cygwin

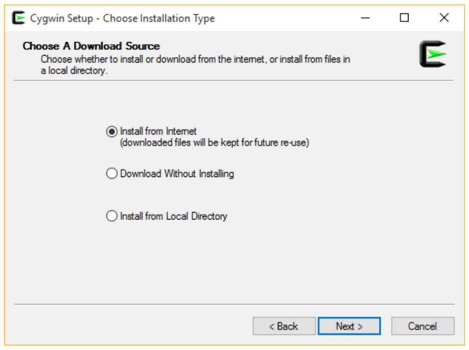
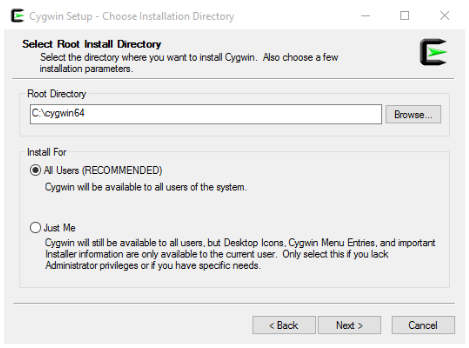
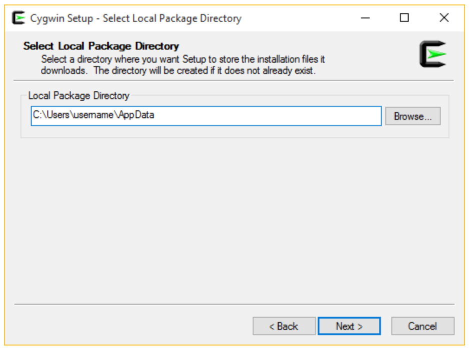
Cygwin is a Linux- like environment for Windows. The other two operating systems that were discussed above were developed with Linux, while Windows was not. So, in order to run applications like Bertini that need Linux, there needs to be a program or environment that provides Linux for the applications that need it.

How to Install Cygwin Cygwin can be found at cygwin.com. Please make sure to choose the version (either 32-bit or 64-bit) that is appropriate for your laptop. After the setup-x86.exe (or setup-x86_64.exe) have downloaded, run the application, and follow the instructions.

Instructions	Screen Shot
Click 'next' on the first screen.	 **Cygwin Net Release Setup Program** This setup program is used for the initial installation of the Cygwin environment as well as all subsequent updates. Make sure to remember where you saved it. The pages that follow will guide you through the installation. Please note that Cygwin consists of a large number of packages spanning a wide variety of purposes. We only install a base set of packages by default. You can always run this program at any time in the future to add, remove, or upgrade packages as necessary. **Setup.exe version 2.871 (64 bit)** Copyright 2000-2015 http://www.cygwin.com/

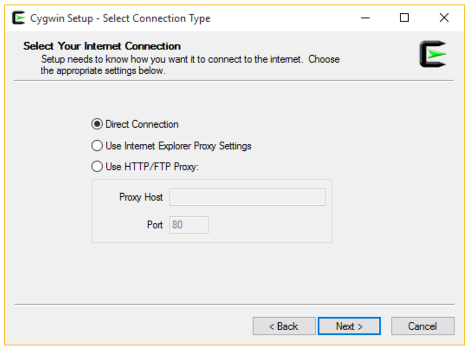
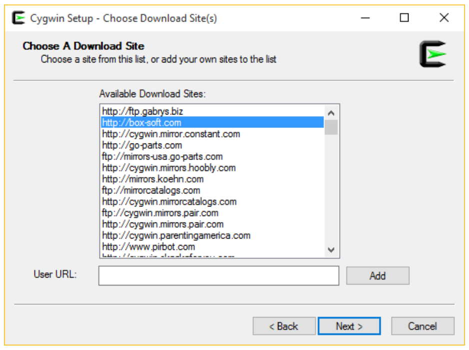
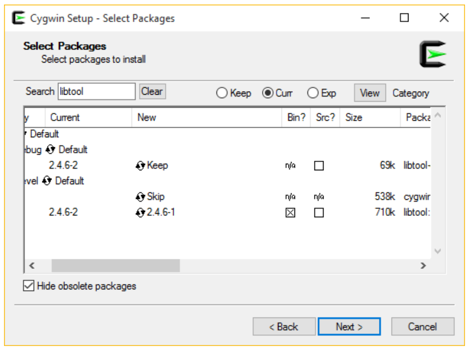
Continued on next page

Table 3: *Continued from previous page*

Instructions	**Screen Shot**
Select the 'Install from Internet' option; click 'next'.	 Cygwin Setup - Choose Installation Type **Choose A Download Source** Choose whether to install or download from the internet, or install from files in a local directory. ● Install from Internet (downloaded files will be kept for future re-use) ○ Download Without Installing ○ Install from Local Directory
Enter the preferred installation directory; click 'next'.	 Cygwin Setup - Choose Installation Directory **Select Root Install Directory** Select the directory where you want to install Cygwin. Also choose a few installation parameters. Root Directory: C:\cygwin64 Install For ● All Users (RECOMMENDED) Cygwin will be available to all users of the system. ○ Just Me Cygwin will still be available to all users, but Desktop Icons, Cygwin Menu Entries, and important Installer information are only available to the current user. Only select this if you lack Administrator privileges or if you have specific needs.
Choose a temporary installation folder; click 'next'.	 Cygwin Setup - Select Local Package Directory **Select Local Package Directory** Select a directory where you want Setup to store the installation files it downloads. The directory will be created if it does not already exist. Local Package Directory: C:\Users\username\AppData

Continued on next page

Table 3: *Continued from previous page*

Instructions	**Screen Shot**
Select the 'Direct Connection' option; click 'next'.	
Choose a download site; click 'next'.	
Select the packages that you will need. , then click 'next'. See below for more instructions	
If during the course of installation, a message pops up and says that certain dependencies are required for the packages, click the 'yes' button. When it has installed everything, select 'finish'.	

Cygwin Setup - Select Connection Type

Select Your Internet Connection
Setup needs to know how you want it to connect to the internet. Choose the appropriate settings below.

[✓ Direct Connection]
Use Internet Explorer Proxy Settings
Use HTTP/FTP Proxy:
Proxy Host
Port 80

< Back | Next > | Cancel

Cygwin Setup - Choose Download Site(s)

Choose A Download Site
Choose a site from this list, or add your own sites to the list

Available Download Sites:
http://ftp.gabrys.biz
[✓ http://box-soft.com]
http://cygwin.mirror.constant.com
http://go-parts.com
ftp://mirrors-usa.go-parts.com
http://cygwin.mirrors.hoobly.com
http://mirrors.koehn.com
ftp://mirrorcatalogs.com
http://cygwin.mirrorcatalogs.com
ftp://cygwin.mirrors.pair.com
http://cygwin.mirrors.pair.com
http://cygwin.parentingamerica.com
http://www.pirbot.com

User URL: | Add

< Back | Next > | Cancel

Cygwin Setup - Select Packages

Select Packages
Select packages to install

Search libtool | Clear | Keep [✓ Curr] Exp | View | Category

y	Current	New	Bin?	Src?	Size	Packa
Default						
bug Default						
	2.4.6-2	Keep	n/a	☐	69k	libtool-
vel Default						
		Skip	n/a	n/a	538k	cygwir
	2.4.6-2	2.4.6-1	☒	☐	710k	libtool:

[✓ Hide obsolete packages]

< Back | Next > | Cancel

Cite Source Here

Selecting packages for Cygwin

The list of packages that you will need for Bertini_real can be found below. To find the packages, a user can type the name into the search in the top left of the menu, which will then show the packages containing that name (e.g. 'libtool'). To choose a package click on the text that says 'Skip' until it changes to a version number (e.g. '2.4.6-1').

autoconf	automake
bash	bison
boost (all the C and C++ libraries)	bzip2
X-11	emacs (or nano or some other text editor that you prefer)
flex	vim
mingw-gcc-g++-4.7.3-1	cygutils-X11
gmp	gcc
mpc	mpfr
libzip2	xinit
libtool	openmpi
openssl	openssh
tar	

Setting up Cygwin

Initializing Paths

In order to properly run Cygwin, you need to add Cygwin to the PATH variable. In order to do so, follow these steps:

1. Open the Control Panel and select 'System'.
2. Select the 'Advanced System Settings', and then the 'Environment Variables' option.
3. In the window that appears, select the system variable "PATH' and append `; C:\cygwin\bin` to the end of the PATH variable. When you are doing this, you can also append `; C:\path\to\matlab.exe` to the end of the PATH as well.

Organizing Cygwin

Once Cygwin and MATLAB have been added to the PATH variable, you are now ready to open and run Cygwin. For users who are not familiar with Cygwin, a good reference sheet can be found here.

As part of the installation process, Cygwin will automatically configure and install the packages you selected. This is useful, since it saves a lot of time for the user. However, this also allows a Cygwin user to go and be able to automatically use some of these applications, such as 'libtoolize'. Libtoolize, one of the packages that was installed with `setup86x.exe` allows a user to set up a shared library format. In other words, a user doesn't have to call each different library; they are already set up and in the same place.

When I set up Cygwin, I created a new folder located in `\usr\local` that would contain any downloaded files from the Internet that would be used with Cygwin.

When setting up Cygwin, I found that in order to install the dependencies that are needed for Bertini and Bertini_real, they had to be downloaded from the Internet. The following instructions describe how to install these dependencies and finish setting up Cygwin, and were paraphrased from How to Install a Newer Version of GCC.

Installing Dependencies

As stated earlier, the dependencies that need to be downloaded are:

C++ compiler	gcc(Cygwin)	✓
MPI	openmpi(Cygwin)	✓
Boost	boost(Cygwin)	✓
GMP	gmp(Cygwin)	✓
MPFR	here	
GNU Multiple Precision Complex Library (MPC)	here	

Once you have downloaded the programs from the sites, put the zipped files in the folder that you created in
`\usr\local\your_folder`. Then, in Cygwin, enter `your_folder`.

Linking Cygwin Environment Paths

After logging into Cygwin, a user needs to set up their environment paths inside the terminal before they set up the files they downloaded. In order to see how the paths currently are set up, either type the following code into the terminal, or copy it and paste it into the terminal:

```
echo ;\
echo LD_LIBRARY_PATH=${LD_LIBRARY_PATH}; \
echo LIBRARY_PATH=${LIBRARY_PATH}; \
echo CPATH=${CPATH}; \
echo PATH=${PATH}; \
echo   \cite{installnewerGCC}
```

File 13: Adapted from [6]

Some things to keep in mind while setting up the environment variables LD_LIBRARY_PATH, LIBRARY_PATH, and CPATH:

- **LD_LIBRARY_PATH** and **LIBRARY_PATH** should contain /usr/local/lib (**LIBRARY_PATH** shall not be set on Enterprise Linux Enterprise Linux Server release 5.5 (cartage))
- **CPATH** should contain /usr/local/include
- If **PATH** contains `c:/windows/system32` (or `/cygdrive/c/windows/system32`; case-insensitive), it should be after /bin and /usr/bin. Otherwise the scripts will try to run Windows sort.exe instead of the Unix command with the same name.

To change or modify the different variables, you can use the code below (or you can change the variables in the Control Panel, as shown earlier):

```
setenv LD_LIBRARY_PATH /usr/local/lib
setenv LIBRARY_PATH /usr/local/lib
setenv CPATH /usr/local/include
```

File 14: Adapted from [6]

However, if Cygwin shows a message such as `-bash:  setenv:  command not found`, then you need to use the code below:

```
   export LD_LIBRARY_PATH=/usr/local/lib
# Depending on system, LIBRARY_PATH shall not be set -
#  export LIBRARY_PATH=
   export LIBRARY_PATH=/usr/local/lib
   export CPATH=/usr/local/include
```

File 15: Adapted from [6]

Building and Installing Packages

Now that the environment variables are set up, we can now build and set these packages.

Perform the following build/install steps for the **MPFR** and **MPC** packages *in that order*:

1. cd to your workspace directory (above, e.g., cd `\usr \local \your_folder`)
2. Extract the tarball using tar (e.g., `tar -xf mpfr-3.1.3.tar.bz2`). This will create a subfolder with the source for the given package cd into that source folder (e.g., `cd mpfr-3.1.3`)
3. Type `libtoolize` into the command line and press enter. This will add the files, once they have been compiled, to the shared library.
4. Generate `configure`, by running the command `autoreconf -i`.
5. Read the README and/or INSTALL file if present
6. Note that for the current version of **mpc (0.9)** there is a change that may need to be made to have the build work successfully. You need to edit the line of "mpc.h"
   `#if defined(__MPC_WITHIN_MPC) && __GMP_LIBGMP_DLL` to `#if defined __GMP_LIBGMP_DLL`
7. run `./configure` (this will check the configuration of your system for the purpose of this package)(you also need specify `--enable-static --disable-shared` when compiling the library)
8. run `make` (this will build the package; `-j` can speed things up here)
9. run `make check` (strongly recommended but optional; this will check that everything is correct)
10. run `make install` (this will install all the relevant files to the relevant directories)
11. run `make clean` (optional; this will erase intermediate files - important if you are re-attempting a broken build!)

A.5 Installing Bertini and Bertini_real

Once all of the dependencies have been installed, now Bertini and Bertini_real can be installed. The zip file for Bertini can be found here, while the download site for Bertini_real can be found there

Move these downloads to your terminal, and unzip and install the two programs. To unpack the directory, just run `tar -zxvf FILE_NAME` into the command line while in the folder that the .tar.gz is located. (For Cygwin users, this means going through the same steps as you did for GMP, MPFR, and MPC.) Be sure to install Bertini *before* Bertini_real!!

A.6 Setting up MATLAB

After you have set up Bertini and Bertini_real, you probably want to be able to see a 3D rendering of your solutions. In order to do so, go to the GitHub Bertini_real site and download the .zip file. I recommend that a new folder is created and that the zip folder goes inside this 'master folder'. Unfortunately, the user also has to download each of the functions that are on the same level as the zip folder- e.g. `dehomogenize.m`, `find_constant_vars.m`, etc. and save those in the 'master folder' as well. Make sure that this 'master folder' is linked to the folder where the Bertini and Bertini_real solutions will be located. Once all of these functions and the zip folder are downloaded and saved, then you will be able to successfully use MATLAB in conjunction with Bertini and Bertini_real. Congratulations, you have now made it through the installation process. There are some additional features that can also be installed if you so desire, as well as a practice run in order to verify that everything installed correctly.

A.6.1 Parallel Bertini_real Installation

Bertini_real is parallel enabled, using MPI. You cannot build Bertini_real without support for MPI at the current time. To use multiple processors to decompose a real object, call Bertini_real as you would any other MPI program: `mpiexec [options] bertini_real`.

A.7 Testrun – the Cayley Cubic

Now that everything has been installed, we can now do a test run, to make sure that everything is working properly. To test this program, we will try to generate a Cayley Cubic using the above programs, following a number of steps. The first step is to create an input file. Open up a text editor in your terminal and create a file called `input`.

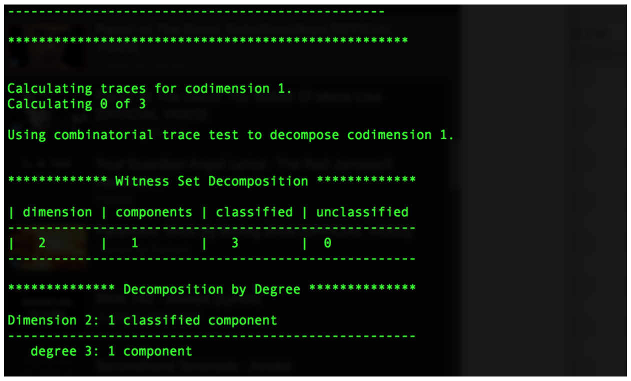
Below is the text for this input file.. A key feature to notice is the second line, where the `tracktype` configuration is set to 1. This configuration setting is necessary for Bertini_real to run.

```
CONFIG
tracktype:1;

END;
INPUT
variable_group x, y, z;
function f;
f = 4 * (x^2 + y^2 + z^2) + 16*x*y*z - 1;
END;
```

File 16: `input` for the Cayley Cubic

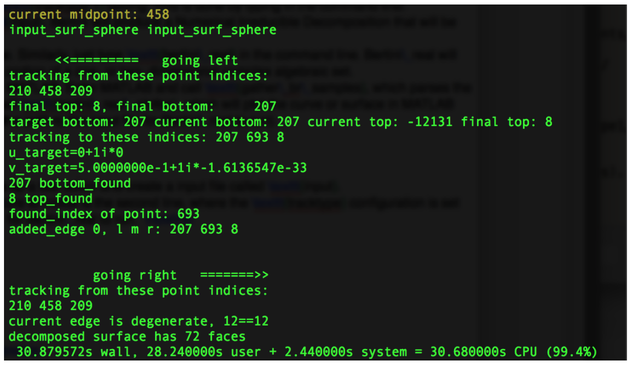
Once the input file is created, we can now run Bertini. Simply navigate in the command line to the directory of the input file and type `bertini` or `bertini input`. You may need to type in the entire pathway to where Bertini is located, if it's not in the same folder, so the command line read

`/cygdrive/path/to/BertiniSource_v1.5/bertini-serial.exe input`
Cygwin users: A user may also use `bertini-serial.exe` (or `bertini_parallel.exe`). This will run Bertini, creating the Numerical Irreducible Decomposition needed for Bertini_real. The following should print to the terminal or shell:



```
-------------------------------------------------

****************************************************


Calculating traces for codimension 1.
Calculating 0 of 3

Using combinatorial trace test to decompose codimension 1.


************* Witness Set Decomposition *************

| dimension | components | classified | unclassified
-----------------------------------------------------
|   2       |   1        |   3        |  0
-----------------------------------------------------

************** Decomposition by Degree **************

Dimension 2: 1 classified component
-----------------------------------------------------
   degree 3: 1 component
```

Once Bertini is finished, the output can be verified as satisfactory (or not). Then, Bertini_real can be run by calling `bertini_real` in the command line. Cygwin users, the same rules that applied to Bertini also apply to Bertini_real, so be sure to include that '.exe' at the end! However, if the input file used was named 'input', no file name is needed at the end of the command line. This program should run for roughly 20-30 seconds, with the final terminal/shell output appearing below:



```
current midpoint: 458
input_surf_sphere input_surf_sphere

      <<==========   going left
tracking from these point indices:
210 458 209
final top: 8, final bottom:     207
target bottom: 207 current bottom: 207 current top: -12131 final top: 8
tracking to these indices: 207 693 8
u_target=0+1i*0
v_target=5.0000000e-1+1i*-1.6136547e-33
207 bottom_found
8 top_found
found_index of point: 693
added_edge 0, l m r: 207 693 8


           going right   =======>>
tracking from these point indices:
210 458 209
current edge is degenerate, 12==12
decomposed surface has 72 faces
 30.879572s wall, 28.240000s user + 2.440000s system = 30.680000s CPU (99.4%)
```

Finally, MATLAB can be used to visualize the result from the Bertini_real run. Open MATLAB and enter the 'master file', which must be linked to the folder where the Bertini_real solutions are located. This can be done by first making sure that you are currently in the 'master folder', then typing `addpath('C:\cygwin64\path\to\solutions_folder')` into the command window and pressing enter. Then you can call `gather_br_samples` in the command window, which generates a .mat file. Then, call `bertini_real_plotter`. This will create a MATLAB figure, pictured below.

If you've been able to reproduce the above figure, then you've mastered the basics of Bertini_real.

A.8 Other Operating Systems

If you would like to use Bertini_real on operating systems that haven't been listed in this manual, please contact Daniel Brake to discuss porting Bertini_real to your system.

B Output Files

In this section, we describe the formats of the plain-text output files from Bertini_real. We document abstractly, and let the user explore concretely by generating data.

By default, the output from Bertini_real is dumped into a folder named `output_dim_D_comp_C`, where `D` and `C` are the dimension and component numbers. This can be overridden by specifying option `-o` at runtime.

B.1 Regardless of dimension

Some files are produced regardless of object dimension:

- `decomp` – B.1.1
- `input_dim_D_comp_C_deflated` – a plain copy from Bertini_real's input in the containing folder.
- `run_metadata` – B.1.2
- `V.vertex` – B.1.3
- `vertex_types` – B.1.4
- `witness_data` – a plain copy from Bertini's output in the containing folder.
- `witness_set` – B.1.5

Additional important files written NOT into the output folder:

- `Dir_Name` – B.1.6

B.1.1 `output/decomp`

```
input_filename
number_vars dimension // number_vars includes homvar

FOR // # is dimension
        number_vars_in_projection
        FOR
                proj_coord_real proj_coord_imag
        END FOR
END FOR

number_patches

FOR // # is number_patches
        number_vars_in_patch
        FOR
                patch_coord_real patch_coord_imag
        END FOR
END FOR

sphere_radius_real sphere_radius_imag
num_sphere_vars  // # is number natural variables
FOR // # is number natural variables
        sphere_center_coord_real sphere_center_coord_imag
END FOR

number_crit_fibers
FOR // # is number_crit_fibers
        crit_fiber_coord_real crit_fiber_coord_imag
END FOR
```

File 17: `output/decomp`

B.1.2 `output/run_metadata`

```
bertini_real_version
/path/to/containing/folder
timing statistics from Boost.Chrono
```

File 18: `output/decomp`

B.1.3 output/V.vertex

```
num_vertices num_projections  num_variables  num_filenames //
    number_vars includes homvar

FOR  // num is num_projections
        FOR // num is num coords incl homvar
                proj_coord_real  proj_coord_imag
        END FOR
END FOR


FOR  // each filename
        num_chars_in_filename
        filename
END FOR

FOR // each vertex
        num_coords // may include synthetic variables
        FOR
                coord_real  coord_imag
        END FOR

        num_projection_values
        FOR
                proj_val_real  proj_val_imag
        END FOR

        input_filename_index
        vertex_type  // see file vertex_types
END FOR
```

File 19: `output/V.vertex`

B.1.4 output/vertex_types

```
num_vertex_types

FOR
        VertexType value
END FOR
```

File 20: `output/vertex_types`

Versions of Bertini_real prior to 1.4 used sequential type indices rather than binary values, to vertex types. We converted to binary indices to allow vertices to have multiple types, and represent this compactly. The Matlab visualization code takes advantage of this, and can plot points in each of the type categories they appear in. The purpose of this file is to let one write code which self-adapts to any future vertex types.

B.1.5 output/witness_set

```
num_points dimension component_number

FOR // each point
        FOR // each variable
                coord_real coord_imag
        END FOR
END FOR

num_linears num_vars
FOR // each linear
        FOR // each coordinate
                linear_coeff_real linear_coeff_imag
        END FOR
END FOR

num_patches num_vars
FOR // each patch
        FOR // each coordinate
                patch_coeff_real patch_coeff_imag
        END FOR
END FOR
```

File 21: `output/witness_set`

B.1.6 /Dir_Name

```
/path/to/output/folder
MPType // why? I don't know, it seemed important early on.
dimension
```

File 22: `output/V.vertex`

B.2 Curve files

There are several files written for every curve decomposed, into the containing folder. For surfaces, each sub-curve is written to its own sub-folder, appropriately named.

- `output/curve.cnums` – B.2.1
- `output/E.edge` – B.2.2

B.2.1 `output/curve.cnums`

This file contains the cycle numbers of the paths, tracked from the generic midpoint to the left and right points. Degenerate edges get 0's because there is no path. Otherwise, the numbers are at least 1.

```
number_edges

FOR // each edge
        going_left going_right
END FOR
```

File 23: `output/curve.cnums`

B.2.2 `output/E.edge`

These are 0-based indices into `V.vertex`.

```
number_edges

FOR // each edge
        left mid right
END FOR
```

File 24: `output/E.edge`

B.3 Surface files

- `output/F.faces` – B.3.1
- `output/S.surf` – B.3.2

B.3.1 `F.faces`

```
number_faces

FOR // each face
        midpoint
        critslice_index
        top_edge_index  bottom_edge_index
        system_name_top  system_name_bottom

        num_left_edges
        FOR // each left edge
                edge_index
        END FOR

        num_right_edges
        FOR // each right edge
                edge_index
        END FOR
END FOR
```

File 25: `output/F.faces`

B.3.2 `S.surf`

```
number_faces  0  num_midslices  num_critslices

number_singular_curves
FOR // each singular curve
        singcurve_multiplicity  index
END FOR
```

File 26: `output/S.surf`

Acronyms

GMP	GNU Multiple Precision Arithmetic Library. 21, 26
MPC	GNU Multiple Precision Complex Library. 26
MPFR	Multiple Precision Floating-Point Reliable. 21, 26
MPI	Message Passing Interface. 21
RGB	Red Green Blue color values. *Glossary:* RGB triple
STL	stereolithography. 16