Real answers to complex questions with numerical nonlinear algebra

Danielle Amethyst Brake [1]

[1] University of Wisconsin - Eau Claire

September 9, 2018

Contents

1 Preface

1.1 A note from an author

The motivation for this guide is the Nonlinear Algebra Bootcamp, at the workshop on Nonlinear Algebra at the International Center for Experimental Research in Mathematics at Brown University, Fall 2018.

These exercises are meant to help train you to use tools in the world of *real* computational nonlinear algebra. They were prepared by a numerically aligned person, so they have that flavor to them. This document does not contain a full set of references, though they are in some places provided. Feedback is welcomed to danielleamethystbrake@gmail.com.

We'll start by solving a few simple problems, and build up to some that are deeper. The challenge problems toward the end are open, credited to the researcher who posed it. They're food for thought and intended as the beginning of ongoing conversations with people here at ICERM this semester.

I hope these problems spur you forward! ♡

1.2 Tools

There are lots of tools in the belt of a computational or applied nonlinear algebraist. Here are some, in $\alpha\beta$ order:

- Alpha certified. command-line.
- Bertini and its friends.
  - Bertini – command-line. Text files as input and output. Interfaces through other languages, including Macaulay2, Matlab,[1] and others.
  - Bertini_real – command-line, visualization in Matlab and Python
  - Paramotopy – command line, visualization in Matlab
  - Bertini2 – command line, Python. experimental
- CoCoa – command line
- Hom4PS – command line
- Macaulay2 – command line, ∃ an online interface.
- Mathematica – proprietary, decent symbolics. Sideways cursors.[2]

[1]From Danielle: if you want to use Matlab for interfacing with Bertini, check out (`git clone`, ha) my mediocrely documented repo of Bertini1 utilities at `https://github.com/ofloveandhate/brakelab`.

[2]Pardon the editorializing. I *am* the author, tho. I think I get to do that.

- Maple – proprietary, good symbolics. Not Danielle's favorite for numerics.
- Matlab – proprietary, oriented well to arrays of numbers. Weaker symbolics.
- PHCPack – command line, online interface

If you want Bertini-flavored input files for any of the Herwig Hauser surfaces, many of them are available at the `test/surface/name` folder in the GitHub repo for Bertini_real.

If you have issues or questions or suggestions, there are many software authors here in the building this semester, so ask around, they're sure to help you!

1.3 Notation

Here are some notes on notation used in this training guide

- I will horrify you with my abuses of notation. Things are vectors when they are, and aren't when they aren't. x is multiple things in one line of math. Yep. This'll get cleaned up over time, as I hope this little booklet, which is mostly just formatting right now, turns into something bigger.
- $\square$ – a placeholder. I also like `_1` and `_2` (those come from C++).
- When in doubt, the system equals zero. Don't make me write $= 0$ all over the place.

1.4 Acknowledgements

In the preparation of this training guide, I had help from many colleagues, including: Jon Hauenstein, Elizabeth Gross, Jeff Sommars, and Anton Leykin.

2 Training exercises

These should be solvable using off-the-shelf software.

2.1 Basic

2.1.1 A simple quadratic

Use as many packages for numerical nonlinear algebra as you can to find the real roots of

$$f(x) = x^2 - 2$$

where in this case we mean, find the x such that $f(x) = 0$.

2.1.2 A simple 2x2 system

Use many numerical softwares for nonlinear algebra to find the real roots of

$$f(x) = \begin{bmatrix} x^3 + xy - 3y \\ xy - 3x^5 + \sqrt{2} \end{bmatrix}$$

2.1.3 A 4x4 system

Describe the real solution set of

$$\begin{bmatrix} (x^2+y^2)^3 - 4x^2y^2(z^2+1) \\ x^4 - 2.5x^2y^3 - xz^3 + y^6 - y^2z^3 \\ 4(\phi^2x^2 - y^2)(\phi^2y^2 - z^2)(\phi^2z^2 - x^2) - (1+2\phi)(x^2+y^2+z^2-w^2)^2w^2 \\ w - 1 \end{bmatrix}$$

where ϕ is the golden number.

What happens as you vary the affine patch – that is, vary the last equation to be another linear function of the variables?

2.2 Intermediate

2.2.1 Roots of cyclic polynomials

Compute the roots of the cyclic-n polynomials,[3] for $n \in \{1, \dots, m\}$ for some practical m. How many are real?

[3] See [1] for a description of the cyclic-n system. Python code to generate the system in a given set of variables is in Appendix A.1

2.2.2 Certification

Certify the reality of the solutions from 2.1.1, 2.1.2, 2.1.3, 2.2.1.

2.2.3 Basic surface plotting

Make a picture of the Cayley Cubic.

What happens in your plot if you center your view tightly around one of the singularities? Do the connected components actually join? At a *point*? How far in can you zoom? What if you change settings? Label the singularity in-figure, and save your picture to disk at ready-to-publish quality.

2.2.4 Varying components on parameters

Describe the way the real components of an elliptic curve depend on its parameters.

3 Deeper problems

3.1 Positive dimension

3.1.1 A real point on every connected component

Compute at least one point on every real connected component of the octic from the paper "A projective surface of degree eight with 168 nodes" [2].[4] How do you deal with the projectivity?

Certify the reality of your solutions.

Challenge: Code the method generically so that you may apply it to any component of any variety with one function call. Document it.

3.1.2 Are there completely naked singularities?

Surfaces often have singularities, and sometimes those singularities are naked – not surrounded in a full-dimension sense by the surface, but instead the local dimension in a neighborhood of some point is 1 or 0, rather than 2.

Consider

- the Whitney Umbrella:

$$x^2 - y^2 z$$

- Buggle:

$$x^4y^2 + y^4x^2 - x^2y^2 + z^6$$

Determine if they have any naked singularities. Deflate the singularities. Are there any singularities on these singularities? If so, deflate them, all the way, so that for every singularity, you have a polynomial system with that singularity as a regular point. Describe the resulting polynomial systems.

3.1.3 An algebraic version of a Möbius strip

We can generate a torus as a surface of revolution with a circle offset from the origin – hey, we've all taken Calc 1. We can generate a flattened torus – a donut with insufficient leavening – by using an ellipse. We can generate a twisty donut by revolving the ellipse about its center one full revolution as the torus is spun about the z-axis. It looks like a Möbius strip, but is definitely not a Möbius strip, so let's call it an algebraic Möbius-like surface.

[4]A Bertini input file for a particular patching of it may be found at `test/surface/dihedral` at Danielle's GitHub repo for Bertini_real.

Is such a Möbius-like surface smooth? What happens in your rendering software if you make the number of twists very large?

4 Challenges and open problems

These problems are written in the first person, in the perspective of the credited mathematician.

4.1 Find a parameter point such that _1?

⋆ From Elizabeth Gross ⋆

This paper (https://arxiv.org/pdf/1502.03188.pdf) focuses on a single parameterized polynomial system arising from a particular signaling pathway model. We know that we can find parameters that give rise to 0, 1, 2, and 3 real positive solutions. We like to challenge people to see if they can find a set of parameters that give rise to more than 3 real positive solutions (See Remark 4.2 in the paper)–so far nobody has been able to. [5]

If people want to experiment with it, the network is one of the pre-loaded examples in the ReactionNetworks.m2 package (http://www2.macaulay2.com/Macaulay2/doc/Macaulay2-1.11/share/doc/Macaulay2/ReactionNetworks/html/_wnt.html).

4.2 Find the posreal solutions

⋆ From Jeff Sommars ⋆

I was contacted by a quantum physics grad student in Sydney who was trying to find positive real solutions to several sets of polynomial systems (I expect they were a parametrizable family, but I couldn't get him to give me a generalized version). I've pasted in the simplest one below in Macaulay2 format. It's a 16x16 system, but the other ones he was interested in were more like 30x30...

```
R = CC[a1,a2,a3,a4,b1,b2,b3,b4,c1,c2,c3,c4,k41,k12,k23,k34]
polys = {b1^2 + 0.2*a1*c1 + c1^2 + a1^2*k41 - a4^2*k41
  , a1*b1*(-0.5 + k41) - a4*b4*k41
  , b2^2 + 0.2*a2*c2 + c2^2 - a1^2*k12 + a2^2*k12
  , a2*b2*(-0.5 + k12) - a1*b1*k12
  , b3^2 + 0.2*a3*c3 + c3^2 - a2^2*k23 + a3^2*k23
  , a3*b3*(-0.5 + k23) - a2*b2*k23
  , b4^2 + 0.2*a4*c4 + c4^2 - a3^2*k34 + a4^2*k34
  , a4*b4*(-0.5 + k34) - a3*b3*k34
  , -1 + a1^2 + b1^2 + c1^2
  , -1 + a2^2 + b2^2 + c2^2
  , -1 + a3^2 + b3^2 + c3^2
  , -1 + a4^2 + b4^2 + c4^2
```

[5] aside from Danielle: I would use Paramotopy's search function to do this...

```
, 0.1*a1^2 - 0.1*b1^2 - 0.1*c1^2 - 1.*a1*c1*(-0.5 + k41) + a4*c4*k41
, 0.1*a2^2 - 0.1*b2^2 - 0.1*c2^2 - 1.*a2*c2*(-0.5 + k12) + a1*c1*k12
, 0.1*a3^2 - 0.1*b3^2 - 0.1*c3^2 - 1.*a3*c3*(-0.5 + k23) + a2*c2*k23
, 0.1*a4^2 - 0.1*b4^2 - 0.1*c4^2 - 1.*a4*c4*(-0.5 + k34) + a3*c3*k34}
```

This is probably out of reach of the `fall18` server. Can you use resources at your home institution? Write to Jeff for the 30x30 system.

4.2.1 How many edges?

⋆ from Danielle Brake ⋆

In a curve decomposition, we break the curve $\mathcal{C}$ into cells called **edges**. A circle always decomposes into two edges – and we get a square.

- What are all possible numbers of edges that a cellular decomposition of some $\mathcal{C}$ may have (depending on the projection used)?
- Is it possible to get other regular polygons, with or without edge-merging? Construct a general method.

4.2.2 Regular polyhedra as raw cellular decompositions

⋆ from Danielle Brake ⋆

In a surface decomposition, we break the surface $\mathcal{S}$ into cells called **faces**. When you decompose a sphere, you always get a regular octohedron – neat! Which of the other Platonic Solids can be achieved as a numerical cell decomposition of some $\mathcal{S}$?

5 Just for fun

5.1 3d printing

5.1.1 Obtain an STL

A `.stl` file, which defines a 3-dimensional triangulation, is 3d printable if there is a well-defined and water-tight interior, as determined by the face normals. Obtain a 3d-printable `.stl` file for

- the surface Leopold:

$$x^2y^2z^2 + 3x^2 + 3y^2 + z^2 - 1$$

- the Whitney Umbrella

$$x^2 - y^2z$$

  How do you deal with the handle?

- the ICERM surface from Cynthia's problem set:[6]

$$\begin{aligned} f =& 3x^2 - x^2y^4 - x^4y^2 - 2x^2z^2 - x^2z^4 - x^4z^2 - 2y^2z^2 - y^2z^4 - \\ & y^4z^2 - 2x^2y^2 - 3x^4 + x^6 + 3y^2 - 3y^4 + y^6 + 3z^2 - 3z^4 + z^6 + 10x^2y^2z^2 - 1 \end{aligned}$$

  bounded inside $[-2, 2]^3$ and below the plane $x + y + z = 1.1$.

- The 4-dimensional surface given by

$$\begin{aligned} f_1 &= w^2 + x^2 + y^2 + z^2 - 1 \\ f_2 &= 2wyz - x(y^2 + z^2) \end{aligned}$$

[6]coefficients here are negated, and the 8 was dropped

A Code

A.1 Cyclic

PyBertini code generating the cyclic-n polynomials

```
def cyclic(vars):
    n = len(vars)
    f = [None] * len(vars) # preallocate
    y = [] #make a circle of variables.  there are more elegant ways to do this
    for ii in range(2):
        for x in vars:
            y.append(x)

    # use a product of a list comprehension to construct the n-1 functions
    for ii in range(n-1):
        f[ii] = numpy.sum( [numpy.prod(y[jj:jj+ii+1]) for jj in range(n)] )

    # the last function is the product of all the variables, minus one
    f[-1] = numpy.prod(vars)-1
    return f
```

References

[1] Yang Dai, Sunyoung Kim, and Masakazu Kojima. Computing all nonsingular solutions of cyclic-n polynomial using polyhedral homotopy continuation methods. *Journal of Computational and Applied Mathematics*, 152(1-2):83–97, 2003.

[2] Stephan Endrass. A projective surface of degree eight with 168 nodes. *arXiv preprint alg-geom/9507011*, 1995.