Samuel Cavender
University of Notre Dame
Department of Applied and Computational Mathematics and Statistics
Spring 2017

bertini_real

Goals

After my first exposure to coding last semester, I began this semester with a broadly-based goal – to explore coding in a mathematical context and better understand its utility. Once I got over the initial hurdle of understanding the structure, functionality, and purpose of the bertini_real program my goal for the semester began to have more direction. As I manipulated and learned the program in the first couple weeks, I came across a problem I felt was worth solving. I had arbitrarily chosen the "buggle" surface to try to decompose using bertini_real, and was confused when I found that it constantly got stuck in isosingular deflation and never finished. My goal for the semester became to get buggle to fully decompose, and do so well enough that it could be 3D printed.

Process

The first step in solving this problem was to understand it more fully. Part of the problem with buggle is its singularities, so I started out by adding a constant to its algebraic input equation in order to eliminate them. With each successful decomposition, I reduced the magnitude of the constant and attempted another. The addition of a constant was disorienting to the surface, even at a mere 1.0E-10, the smallest constant with which I was able to achieve a full decomposition. This eliminated the possibility that this was merely an issue of singularity.

The next step to solving this problem involved learning about how bertini_real's current implementation causes matrices grow combinatorially with each isosingular deflation. We discussed how the file sizes balloon, and how you end up with too many equations that are evaluated repeatedly. When it comes to a surface like buggle that requires many isosingular deflations, the equations become too numerous to evaluate in every deflation, which is why bertini_real always eventually stalls out. However, we realized that if we wrote the surface's algebraic equation as a subfunction rather than a function, and then set the function equal to its corresponding subfunction, then substitution during each isosingular deflation would be prevented. This would allow each equation to be evaluated a single time, and then subsequently used in new computations rather than being evaluated repetitively. This lead to the implementation of the deflate_no_subst function for bertini_real. I wrote a function that identifies whether a given function in an input file is already defined as a subfunction, and then if it isn't proceeds to define it as a subfunction. It will then set the original function equal to the corresponding subfunction. Once I worked out the code in Matlab, I wrote the code in C++ that generates a file containing the information that must be passed to deflate_no_subst and calls the function.

After that, in order to be thorough, we sought to prevent substitution throughout the entirety of the program. The other place where this became an issue was in the computation of the critical curve. A lot of the Matlab code retained the same structure that it had for the deflate_no_subst function, however in this case there weren't multiple functions to work with, and a different set of information was required for the calculation. I identified what modifications needed to be made in order to make the function suitable for computing the critical curve, and implemented the crit_no_subst function. Then, I wrote the C++ code that generates a file containing the information that must be passed to crit_no_subst and calls the function. While the implementation of these functions did improve the computational efficiency of bertini_real, it was not sufficient for the decomposition of buggle. However, it will likely allow for the decomposition of other surfaces that were previously impossible.

Next Step

If I were to keep working on the bertini_real program, and specifically the buggle issue, the next step would be to make it more flexible. I would want to modify the program so that it allows the user to calculate things by hand and pass them to the program at a given point. While the buggle surface cannot currently be deflated with bertini_real, the equations can certainly be worked out by hand. If we could pass those equations to the program and then have it do the rest of the work, we would expect to be able to get a clean, printable rendering of the surface.

Learning Points

Debugging:

Throughout this process, one of the things I think I improved upon most was debugging. Reading and understanding error messages is something that I am constantly improving upon, and getting more experience in Matlab this semester allowed me to expose myself to error messages that I was less familiar with. As a result, I improve my error readings skills as a whole. Working through the debugging process really drove home that a big part of coding is continual improvement, and that collaborative problem solving is an extremely important part of the process.

Metaprogramming:

I had no knowledge of metaprogramming prior to this semester, and it was difficult to wrap my head around the concept initially. Then, even once I understood it, I found that implementing code that does it correctly was even more challenging. Through continual trial and error, I gained a better understanding of what metaprogramming really is and can be. I think the idea of using code to generate code is a wonderful concept, and am fascinated by its creative possibilities.

The Use of Code in Mathematics:

Exploring the use and utility of code in mathematical applications was my broad goal for the semester, and I definitely accomplished it. I got to learn the mathematical concepts that allow bertini_real to function, and then turn around and see how they are implemented in the program. It creates a lot of efficiencies for the user, but there are also lingering inefficiencies, which is why there is always more work to be done. I'm glad that I had the opportunity to help tackle one of those inefficiencies and make the program better for the end user.

Version control:

I found learning version control both difficult, helpful, and insightful all at once. Learning to use Git was difficult because of the total confusion I experienced for most of the time I used it. It was also helpful due to its function, allowing me to work on another person's code. And lastly, it was insightful because I found the way in which it allows you to structure a project extremely powerful. I think that project management skills that I learned as I learned version control will be extremely useful to me in the future, even outside of coding.