Algorithmic Algebraic Geometry REU at Texas A&M: Coamoeba Project Writeup

Jeff Sommars

October 26, 2011

Abstract

For my project as part of the Texas A&M NSF sponsored Research Experience for Undergraduates in Mathematics, I chose to work with coamoebae. Coamoebae were defined for the first time within the past ten years and since then, much fruitful research has been done on them. Two years ago, Lisa Nilsson and Mikael Passare wrote a paper that carefully describes an algorithm for drawing specific cases of two dimensional coamoebae. I developed a program in Sage that takes an A matrix as input from a user and returns the corresponding coamoeba, with several options for ways the user can adapt the coamoeba graph returned. This program has the potential to aid future mathematical research that seeks to gain new insight into coamoebae.

1 Nilsson and Passare's Algorithm

Nilsson and Passare cowrote a paper to find the coamoeba of an A-discriminant. They ingeniously developed a straightforward algorithm that requires relatively few steps to finish constructing the polynomial's corresponding coamoeba. From an A matrix input, the algorithm starts by creating a related B matrix through the use of a Gale transform; immediately after its creation, the B matrix is adjusted. Calculate β_j for each row vector b_j in the B matrix where $\beta_j = -b_{j1}/b_{j2}$. If it is necessary, reorder the b_j as follows:

$$\infty > \beta_1 \geq \beta_2 \geq \ldots \geq \beta_N \geq (-)\infty\,.$$

with β_1 as the top row, β_2 as the second row and so on.

Using this new B matrix, find the Horn-Kapranov parameterization of the B matrix. Though sometimes defined using partial derivatives, an equivalent way to define it is simply by using the formula below and the rows of the B matrix.

$$\Psi[t_1 : t_2] = \Big(\prod_{j=1,2,\ldots,N} < b_j, t >^{b_{j1}}, \prod_{j=1,2,\ldots,N} < b_j, t >^{b_{j2}} \Big)$$

Once parameterized, pick a value of t greater than the greatest b_j or smaller than the smallest b_j and input this t into the Horn-Kapranov parameterization. The crucial information that must be taken is the sign of the ordered pair that this produces. There are four distinct possibilities: $(+,+)$, $(+,-)$, $(-,+)$ and

$(-,-)$. Each one corresponds to a starting point p_0 of the coamoeba: $(0,0)$, $(0,\pi)$, $(\pi,0)$ and (π,π) respectively.

Next, take π times the row vectors of the B matrix and connect them tip to tail, starting at the p_0 found from the Horn-Kapranov parameterization. Though one can start with any row in the B matrix, what is important is that the next vector taken is always one row below the most recently drawn vector, with β_1 considered to be "directly below" β_N. After each row vector has been drawn, the last line segment drawn will always connect back to p_0, though it is quite possible that the polygon drawn will be complex rather than simple. To finish the algorithm, do exactly as before, but instead of multiplying by π, multiply each of the row vectors by $-\pi$. The drawing that results from this process is the principle coamoeba, which consists of two distinct cycles, Γ^+ (the first cycle drawn) and Γ^- (the second cycle).

Nilsson and Passare do a great job rigorously proving each aspect of this algorithm, so I will not take time to address that here (see [1]). Instead, I will remark on a specific challenge that I experienced when coding an implementation of this algorithm.

2 Algorithm Implementation

If each principle coamoeba were a simple polygon that did not intersect any other coamoeba, this project would have taken me two days of programming to complete. However, for nearly every non-trivial A-discriminant, the coamoeba falls into at least one of three categories:

1. The coamoeba drawn in one fundamental domain overlaps a coamoeba drawn in another fundamental domain.

2. The cycle Γ^+ of the principle coamoeba overlaps the cycle Γ^- of the principle coamoeba.

3. The cycle Γ^+ overlaps itself and/or Γ^- overlaps itself.

For the program to draw the coamoeba in a fundamental domain, it must deal with each of these cases, or it will fail to shade each section of the coamoeba properly. There are a variety of ways that all of these problems can be dealt with, some easier to program than others. I chose a way to implement the algorithm that is without question inefficient, but it is a way that can easily be seen to correct all possible errors.

I start by constructing the principle coamoeba and finding the x value and y value of greatest magnitude that are reached by it. I take the absolute value of each of these values and add them to themselves $mod(2\pi)$; call these x_{max} and y_{max}. I then take each ordered pair

$$(-x_{max}, -y_{max}), (-x_{max}+2\pi, -y_{max}), ...(x_{max}, -y_{max})$$

$$(-x_{max}, -y_{max}+2\pi), (-x_{max}+2\pi, -y_{max}+2\pi), ...(x_{max}, -y_{max}+2\pi)$$

$$...$$

$$(-x_{max}, y_{max}), (-x_{max}+2\pi, y_{max}), ...(x_{max}, y_{max})$$

and draw the principle coamoeba at each of them. This will undoubtedly construct extraneous coamoebae, but it will certainly draw each coamoeba that intersects the fundamental domain centered at (0,0). Implementing this deals with the first coloring issue.

Through my algorithm, the final two coloring problems can be handled together. Consider again the principle coamoeba. Take the convex hull, and then triangulate the figure in such a way that none of the lines created through triangulation intersect any existing lines and that no new vertices are created. This creates a set of triangles, some of which are likely to have multiplicity zero, but each having the same multiplicity throughout the entire triangle. I then take a point inside each triangle, test what the multiplicity is by using a winding number algorithm, and then shade the triangle the appropriate shade. Once this is done for each triangle that makes up the principle coamoeba, it corrects for cycles overlapping themselves and for Γ^+ overlapping Γ^-.

Finally, draw a coamoeba using each starting point specified in the previously listed set of ordered pairs, and for each coamoeba drawn, utilize the algorithm that corrects for overlap. Finish by restricting the display of the graph to being only the fundamental domain. This will correct all three coloring issues, although it has the possibility to be highly inefficient.

3 Concluding Remarks

With an implemention as described above, the most time consuming parts of the program are creating the $(2(x_{max})+1)(2(y_{max})+1)$ distinct coamoebae and finding the points within each of the many triangles. An ideal method would eliminate both of these steps. I have explored the possibility of drawing the principle coamoeba in the fundamental domain and every time the coamoeba reaches a boundary, restarting the line segment on the parallel boundary. Though relatively easy to construct the outline of the coamoeba, shading properly becomes an issue that is harder to manage. Triangulating a graph like this is actually quite easy, however a host of new problems immediately arise that I cannot yet resolve. However, if resolved, this could lead to a greatly improved coamoeba drawing program.

As a final thought, I believe there is some connection between the m_B that is defined in the last several pages by [1] and the multiplicity of the distinct regions in the fundamental domain. To be specific, I think that π^{-2} multiplied by the area of the coamoeba's zonotope and the multiplicities in each region of the coamoeba are somehow related. This is something which I have begun to think over the past few days, so I have not had sufficient time to formulate a real hypothesis, though I think the connection could prove useful for efficiently drawing the fundamental domain. Perhaps if someone explores a similar coamoeba project in the future, this possibility can be investigated and the program can be improved.

References

[1] L. Nilsson, and M. Passare, Discriminant coamoebas in dimension two, 2009