Complex Homotopy

Samuel Doud

April 27, 2016

1 Introduction

Complex Homotopy is a simple *Python* application that maps complex numbers from to the complex plane to the complex plane. Formally,

$$T : \mathbb{C} \Rightarrow \mathbb{C}.$$

Throughout this documentation, I will refer to the complex numbers as $\mathbb{C}$. A complex number is a mixture of imaginary and real numbers. The imaginary number i is equal to $\sqrt{-1}$. It can be helpful to imagine the complex numbers as a plane, much like the $xy-$plane, however, on the $xy-$plane we are visualing a function. On the complex plane, we are simply visualizing a set. That is were the use of this program comes in. Observe the function

$$y = x^2$$

Now, observe the same function on the complex "unit grid" and "unit circle".

$$f(z) = z^2$$

Notice how two images were needed for the complex function. With the xy, we can intuitively see the result of the function from the graph, but the complex function is not so easy. In fact, squaring a complex number squares its absolute value and increases its angle measure two-fold. But how would you know that other than analyzing the function?
This program animates the image of z onto $f(z)$ so it is easier for students to observe what is actually occurring with the complex function. Hopefully, it can become as intuitive as any xy function. a. An Overview of your project b. A description of your successes (i.e., what did you accomplish). c. Obstacles and changes of directions (i.e., how did your project change form over the semester, based on what you experienced) d. What did you learn? e. Future directions (i.e., if you had one more month?).

2 Successes

The inital goal of this project was to create a system in which a user could watch a set of points be operated on by a complexz function over time. In that sense,

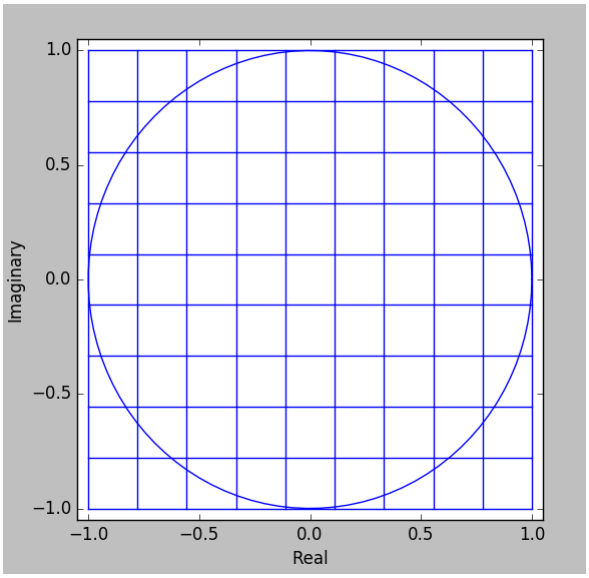

Figure 1: z

this project was highly successful. This system is in place and mostly works. There are some features that you would see on a graphing calculator that are not yet implemented, but they are in the pipeline and more importantly, they are secondary to the actual graphing.

3 Obstacles and changes

4 What I Learned

5 Future Directions

It is likely that I will in fact have another month to work on this project. There is an ever-growing list of items I'd like to implement.

- A decent GUI I have almost never done any work with GUIs. As I spent nearly all my time getting features to run properly on the graphing side, I neglected the graphical layout of the application. This would be the number 1 priority of mine if I had more time. You don't seek out and use applications that look like mine. They look, and are, poorly designed.
- A web interface/API
  The structure for this is already in place as the graphing class and the GUI class of this application are already separate. I would like to see a simple

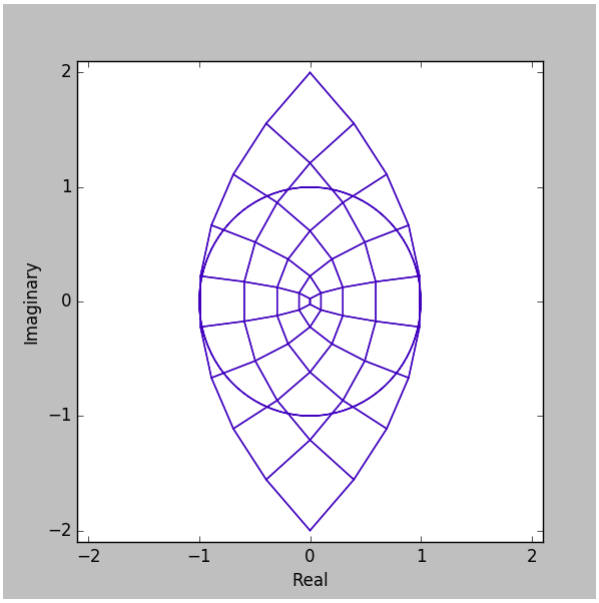

Figure 2: z^2

web app to display these graphs as well as an API for batch requests of graphs.

- A better system for handling division by zero. As it currently stands, if a point is a *singularity*, it is ejected from the computation and a fixed number of points "close" to that are added. However, there are many problems surrounding this. Since we are creating new points, should they be considered in figuring out the limits of the graph? What if these new points are themselves singularities (we could then run into the problem of infinite recursion)?

- A more intuitive way to remove items from the graph.
  As it currently stands, each shape or object is assigned an ID which allows the object to be removed from the graph. However, the interface to which performs this is very rudimentary.

- Multi-layered functions
  This would allow the user to specify to specify two or more functions to operate sequentially. For example, $f(z) = z**2$ and $g(z) = z + 1$. If the user ran $g(f(z))$ they would first see the function f operate, then see the result of that applied to the function $g(z)$. This could be much more clear than just $f(z) = z**2 + 1$.

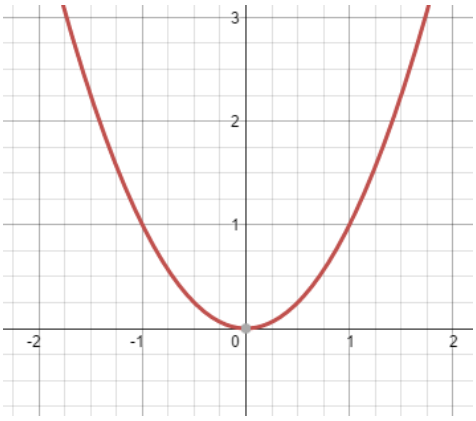

Figure 3: $y = x^2$

6 Technical

6.1 Libraries

The third party libraries that are utilized by Complex Homotopy

6.1.1 sympy

Sympy is a symbolic algebra library for *Python*. It can solve systems of equations, simplify polynomials, and just about anything else algebraic. The use of *sympy* in *Complex Homotopy* is that it allows for the interpretation of complex functions. For example, $f(z) = z^2$; *sympy* can evaluate this function as if it were a complex function and it can handle it symbolically.
If I were to be passed the function $\frac{z^2+1}{2}$ I could "shunt" it to break the function down into steps:

1. Square it
2. Add one
3. Divide by two

If this was implemented, then there would be no need to use *sympy*. However, *sympy* is a tested library that is much more robust than anything I could hope to write. The "Shunting Yard" algorithim is a very diffucult one and opens a pathway for bugs to enter the program. Additionally, *sympy* can make use of *numpy*'s ability to "drop-down" to *C* and run calculations on the SIMD processors of your CPU, enabling much faster computation.

6.1.2 matplotlib

This class is the standard Python graphing library. This forms an abstraction for the xy-plane to the users monitor. For example, $(100, 100)$ on the xy-plane could export to any point on the user's window but will remain in the same position relative to the origin of the graph as the user moves the window. Additionally, matplotlib can handle animation, which is obviously pivotal in this program. Within the animation there is the ability to export the animation as a video file. This functionality requires *ffmpeg* and is therefore is difficult to distribute this feature.

6.2 numpy

Numpy allows for fast calculation of ranges of numbers. Typically in *Python* we would use the built-in *range(n)* function. However, this function only works with positive integers. On the other hand, *numpy* has a *linspace* and *arrange* function that will operate with complex numbers as inputs. For example, if I write

$$\text{numpy.linspace}(0 + 0j,\ 1 + 1j,\ 50)$$

I will get 50 complex numbers between $0 + 0j$ and $1 + 1j$.