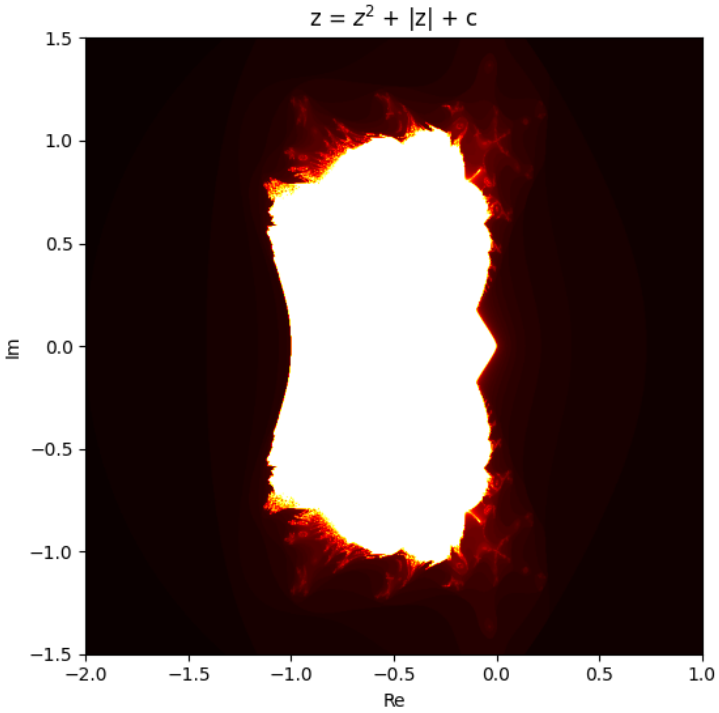
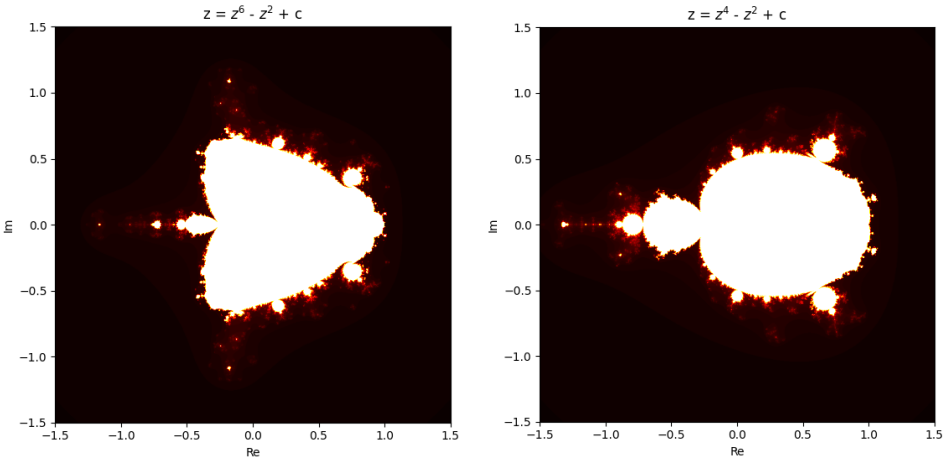
This document is not formatted well. enjoy :)

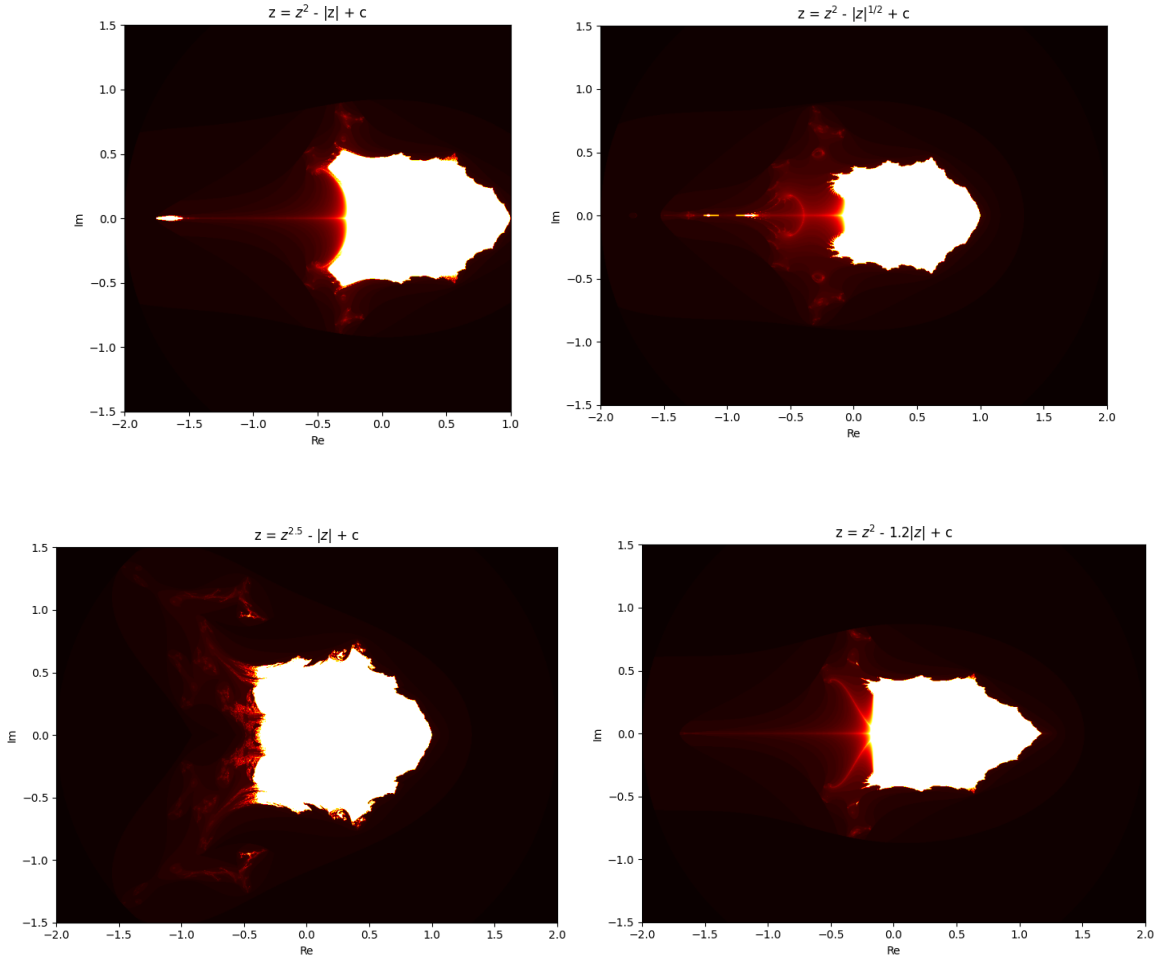
Mandelbrot type fractals. This first set are generated in a process similar to how the Mandelbrot set is generated, $z_{n+1} = f(z_n)$. For example, if $f(z_n) = z_n^2 + |z_n| + c$, you get the following set.



Where c is the complex number defined at each point on the plane. The gradient of white to red to black indicates the amount of time to diverge. So, white areas converge (at least after however many iterations I did), black diverges very quickly and shades of red are somewhere in between. From this process I mucked about with f(z) to create some cool looking fractals.

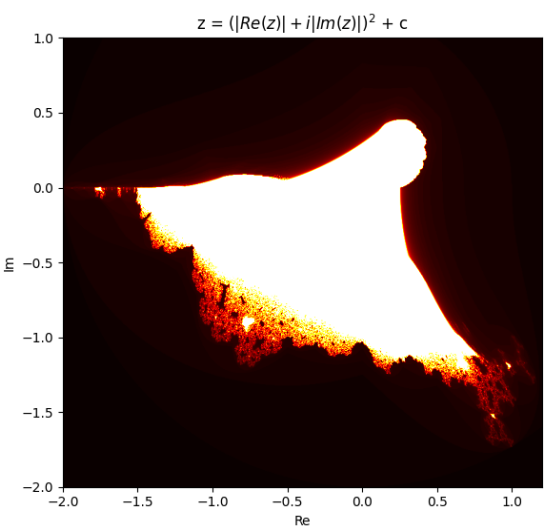


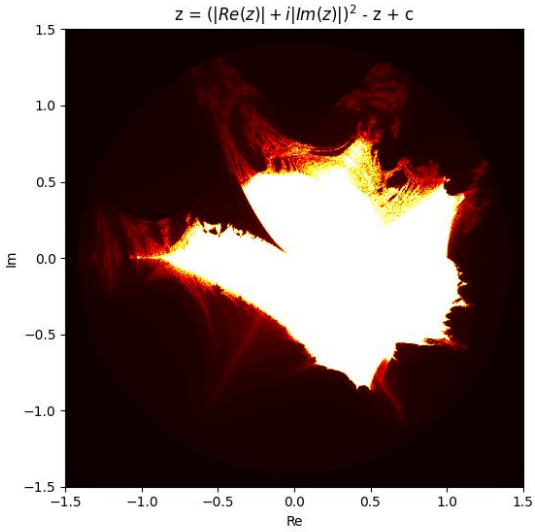
These two are very “mandelbrot-ty” and you can see the familiar bulbs and self-similarity which is present in the mandelbrot set. After this I tried messing about with absolute values.

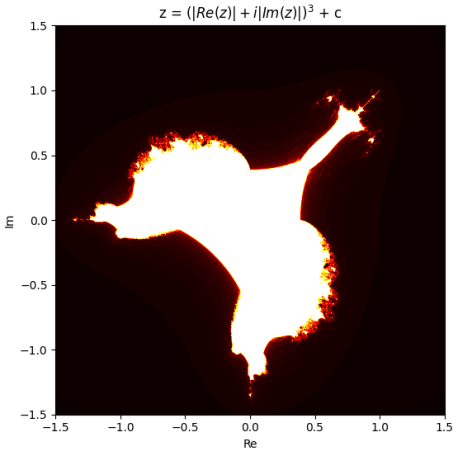


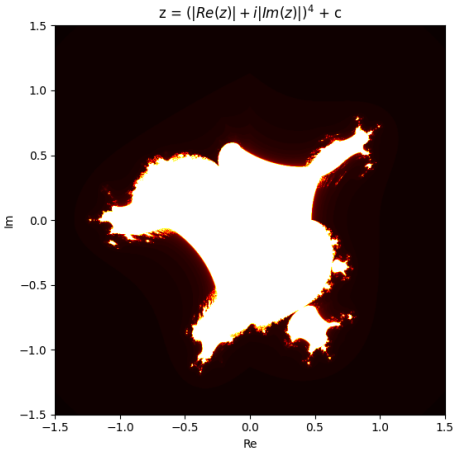
I like the top right one cos of the intracacies between real values of -0.5 and 0, I call that blowing O’s because of the O shapes which look like smoke rings. Also the bottom right one cos it looks like a rhino beetle and hence I’ve named it the rhino beetle. I have no idea if someone else has found these before me and I didn’t really look to be honest, but whatever I am claiming them (at least for now, would be happy to see someone elses).

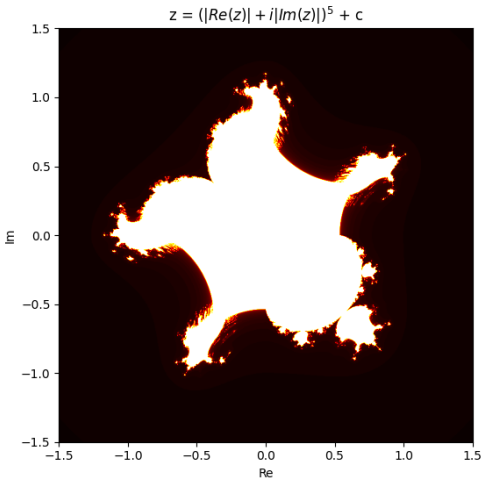
I started using *f(z)*’s which utilised the real and imaginary parts independently. A classic example is the burning ship fractal, pictured top left below. This gives rise to assymetric fractals. For these plots I was just making shit up and seeing what I thought looked cool, did notice some interesting patterns though.

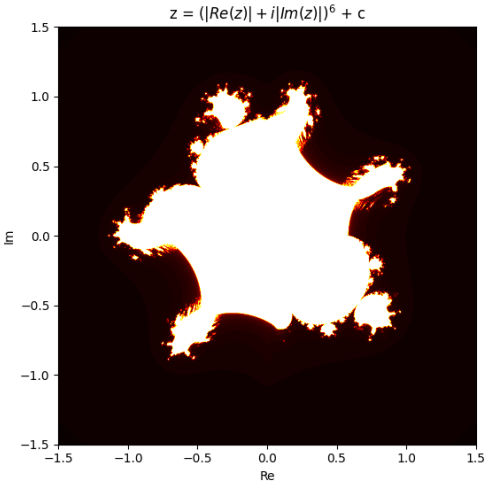




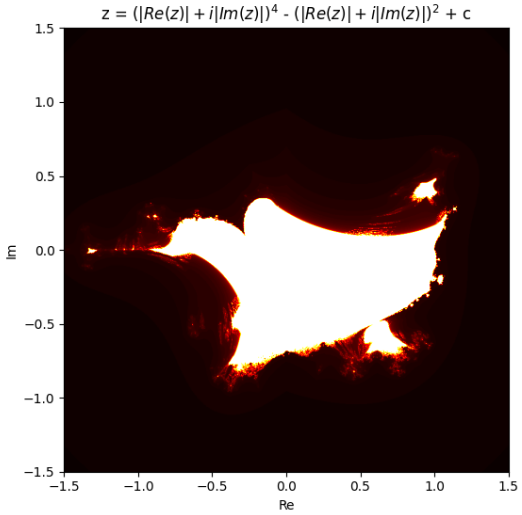




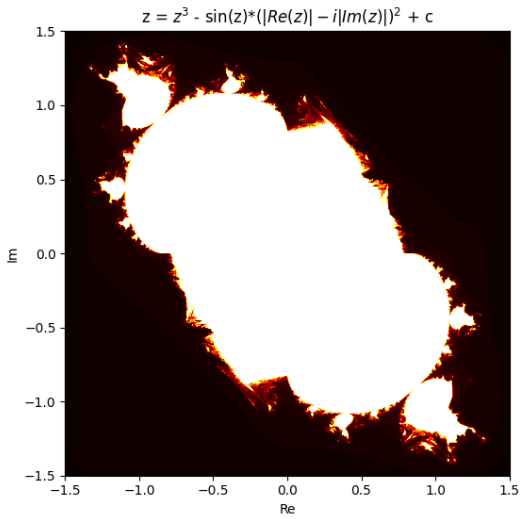


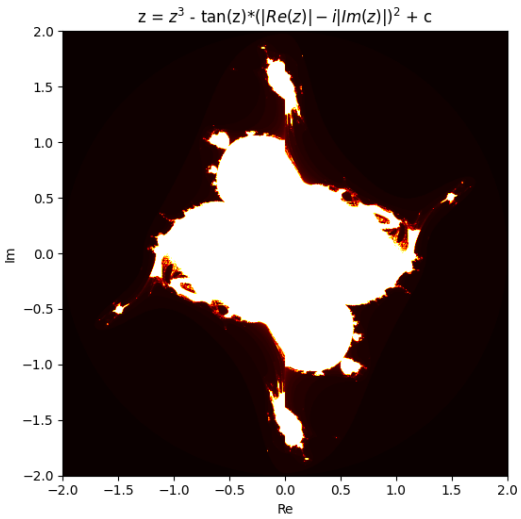


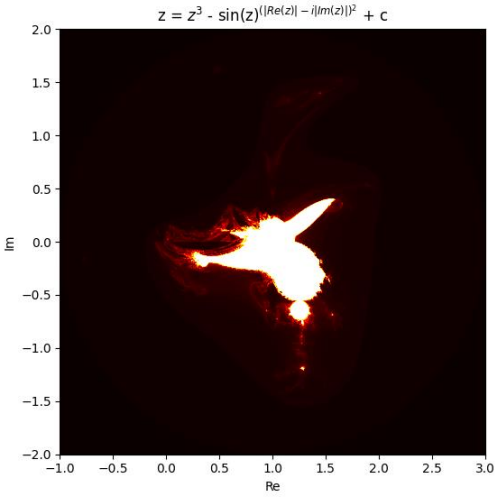
In the above 4 plots, each time the power increases by 1 the fractal “grows another limb” and shifts between symmetric and assymetric fractals for odd and even powers respectively.

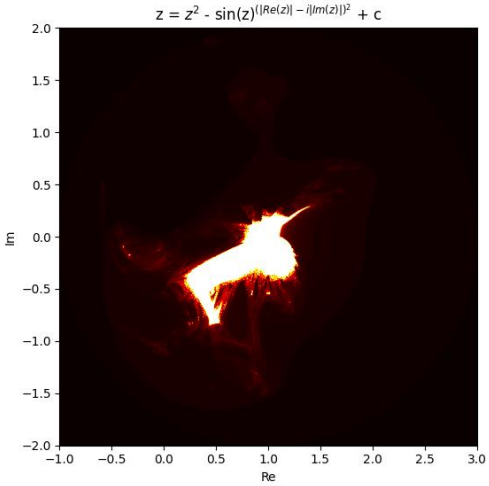


After this I started adding sins, raising stuff to the power of z and other stuff to get even more wild results.









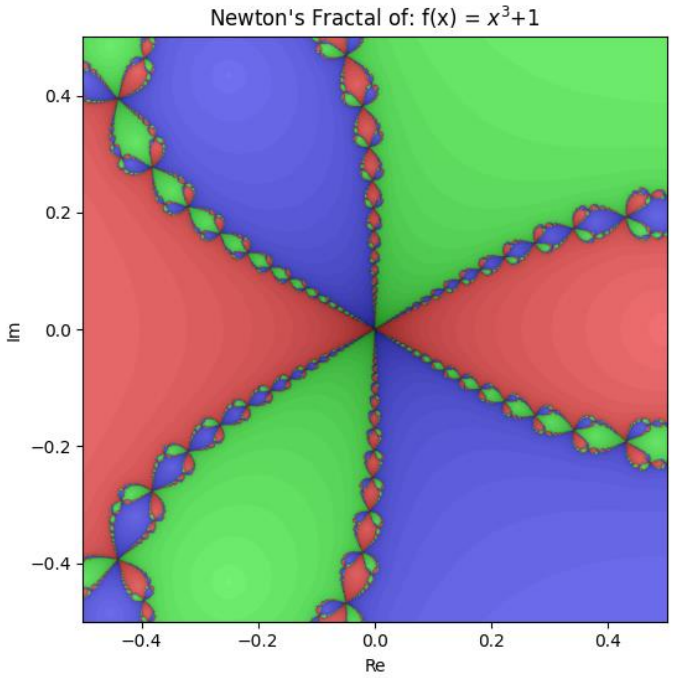
It was after a couple of these I started to question what I was actually doing. Do these fractals show examples of self-similarity? What the fuck is actually going on here? Is there even a way to decipher this in my feeble little brain? I quickly concluded I have no idea and moved on to a different type of fractal, maybe I will return and look at this again one day, but not now.

Newton fractals

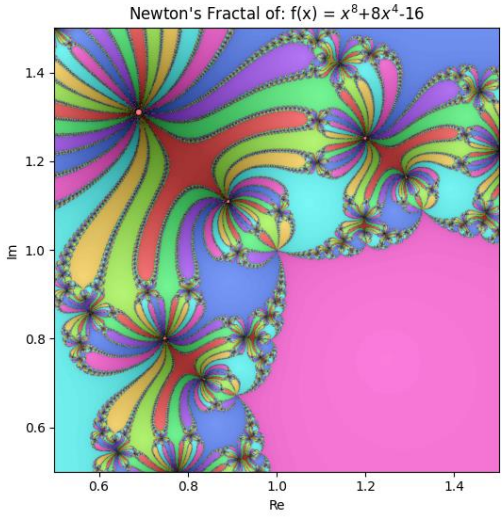
Next I started messing with Newton fractals. These are generated by applying the Newton-Raphson method to a complex plane of points and evaluating which root each point corresponds to. The Newton-Raphson method for a function *f* is as follows:

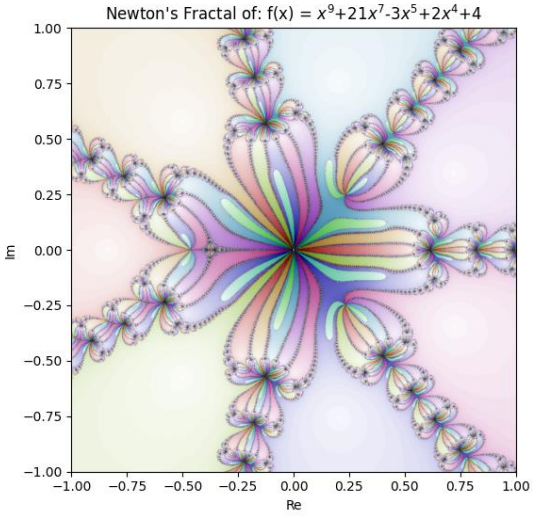
$$Z_{n+1} = z_n - f(z_n)/f'(z_n)$$

Until the point is sufficiently close to one of the roots. A classic exmaple is for the function cube roots of unity, $x^3 = 1$. Below I actually show $x^3 = -1$, but it is the same just flipped over x=0.



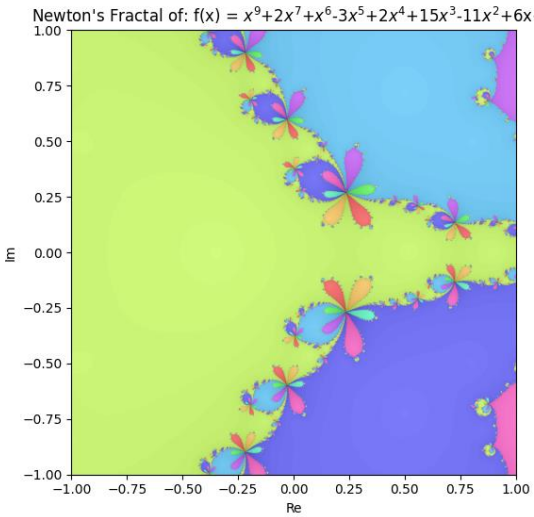
Lighter shades indicate the point converged to a root quickly. 3blue1brown (the G.O.A.T.) did an awesome video on this highly recommend if you have not seen it. But it can be applied to any polynomial.

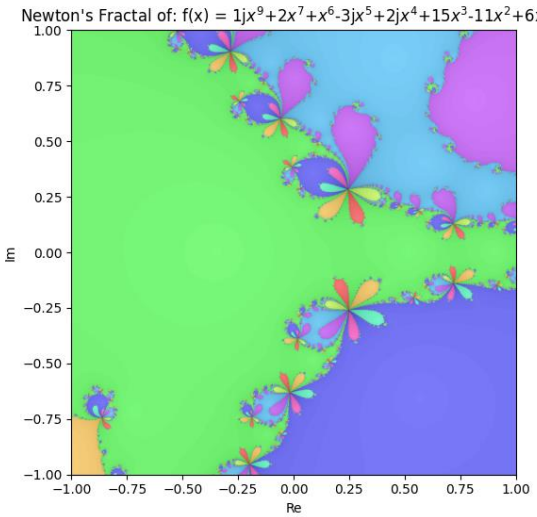




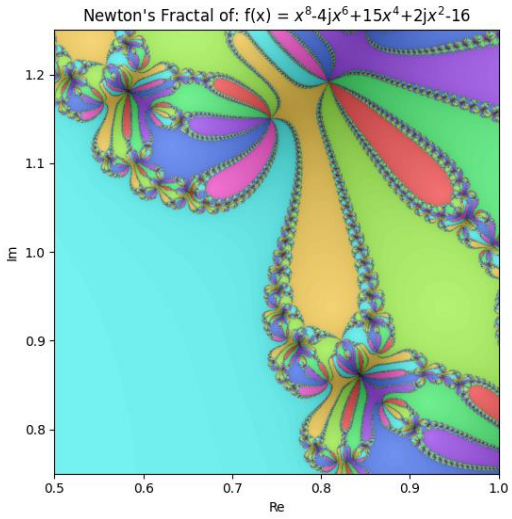
I was messing with some colour schemes Im not sure which one I like more, left shows the roots clearly, while right shows how long it takes to get there more clearly. The red circles in the left image are areas where the iteration limit was reached :(but I couldn't be bothered to improve.

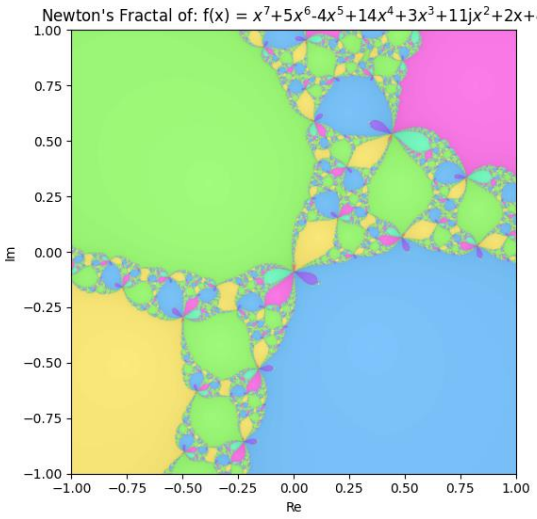
The fractals often (I don't know if always) have some sort of symmetry if only real coefficients are used. However, using complex coefficients distorts this. See below 2 fractals with same coefficients but some of the right images coefficients and complex instead.

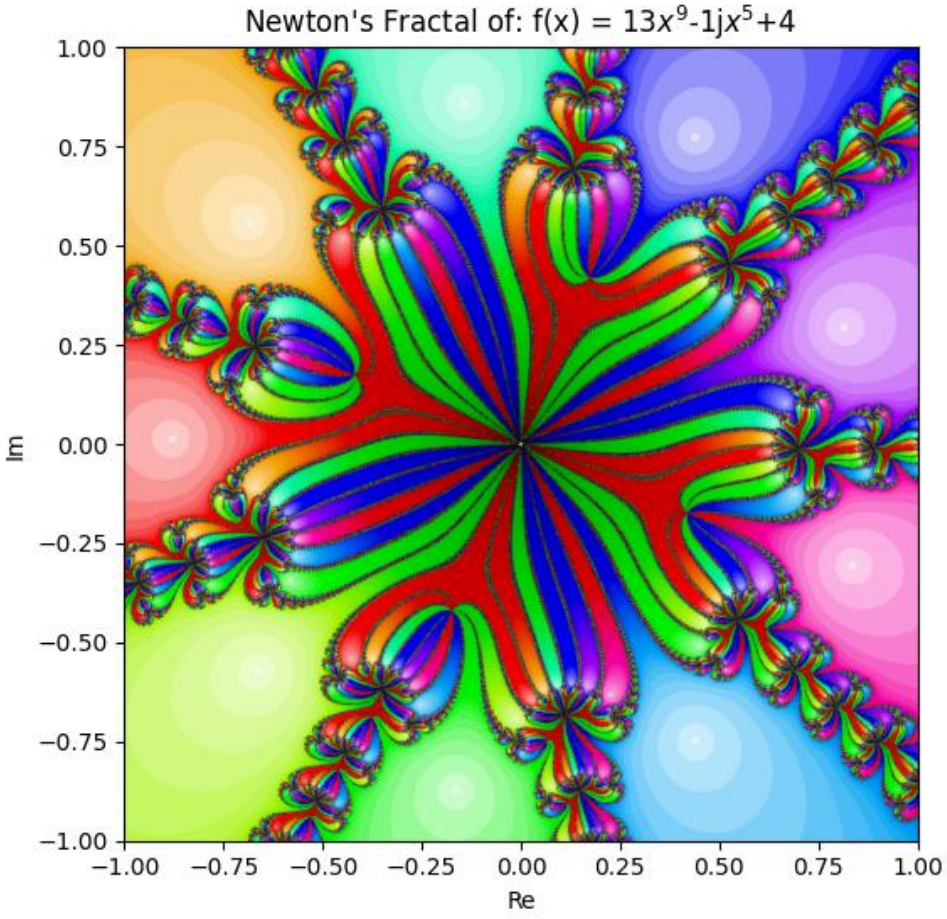




Notice the symmetry across y=0 on the left and the distortion on the right. Most of these examples I pulled the coefficients out of my ass, no rhyme or reason but still look cool.







I just reckon it so mad that you can adjust the coefficients of a polynomial and end with radically different fractals from this simple algorithm.

What more can I explore? What other types of functions can you apply newtons method too? Is it just limited to polynomials? **– research more – extend to f(x) = x^2 + x^-1 + c type polynomials**

Lyapunov Fractals

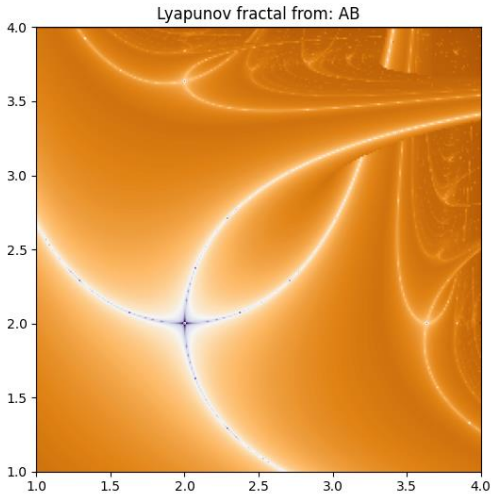
Lyapunov fractals are based off the logistic map, given by: $z_{n+1} = rz_n(1-z_n)$. However, rather then keeping r constant, you pick r based on some sequence S. For example, say S = r_1,r2, then you calculate one step of the map with r_1 and then one step with r_2 and repeat. For simplicity this sequence is often written as S = AB where A means use r_1 and B means use r_2. From this you calculate the Lyapunov exponent of the sequence which determines if a system is chaotic or stable. It is calculated as follows:

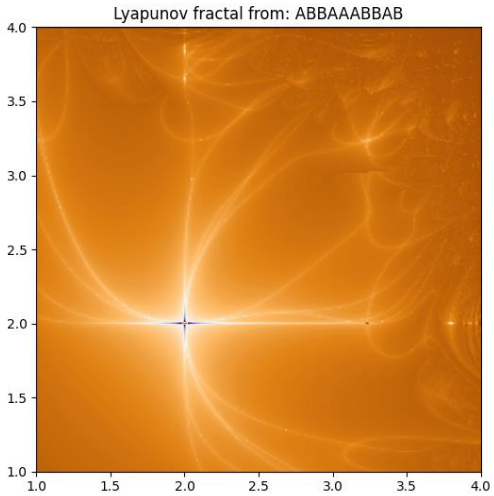
$$\lambda = \lim_{N\to\infty} \frac{1}{N} \sum_{n=1}^{N} \log \left| \frac{dz_{n+1}}{dz_n} \right|$$

Which for the logistic map is:

$$\lambda = \lim_{N\to\infty} \frac{1}{N} \sum_{n=1}^{N} \log |r\,(1 - 2z_n)|$$

And I used N = 10^4 to approximate in my plots. I am not going to go into more detail but I found this https://plus.maths.org/content/extracting-beauty-chaos helpful. The final step is too solve the Lyapunov exponent for every point in a plane (generally around 2 to 4 as this is the interesting region of space) for a given sequence and you've got a Lyapunov fractal. These take a long time to render with my shitty code and old PC so don't expect greatness :|

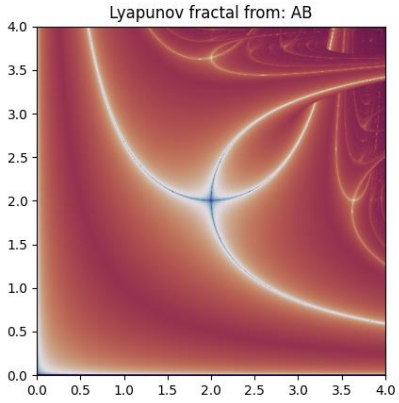


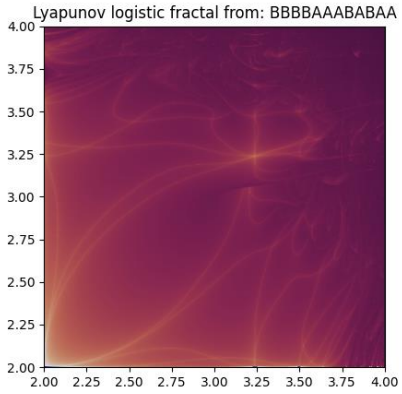


First attempts weren't terrible, this is with 250x250 resolution so not the greatest quality. And I messed up the colouring scheme the chaotic regions are hard to

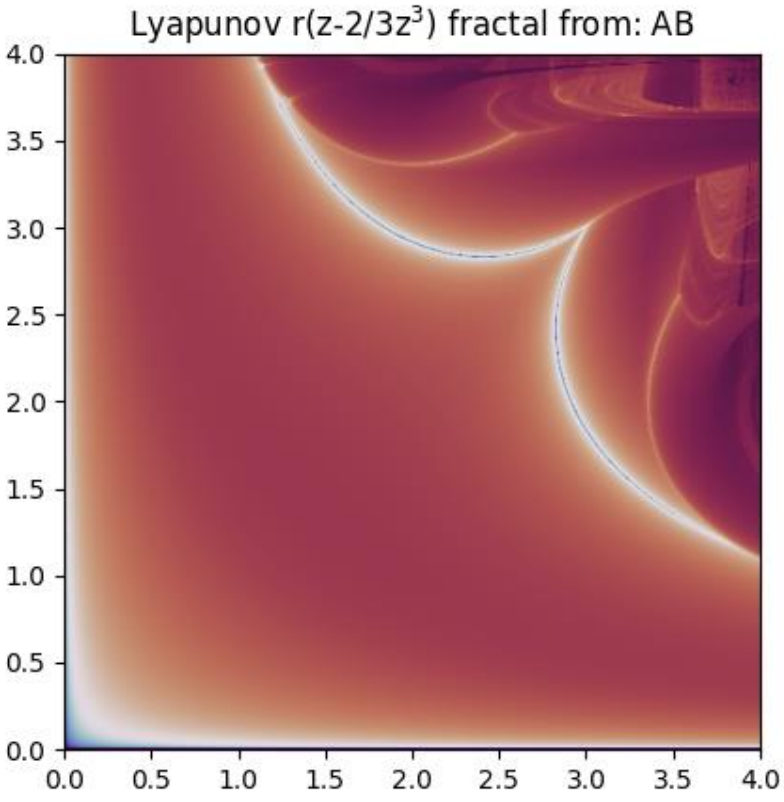
differentiate. Will return and fix in future, but for now I am going to look at other more easily computable fractals. Maybe I will try write this is a better language like C++ could motivate to learn it faster.

Next day: I lowered the amount of iterations to determine the lyapunov expononent. May reduce accuracy slightly but increase in speed makes it worth it. Here are two more better resolution logistic map plots.





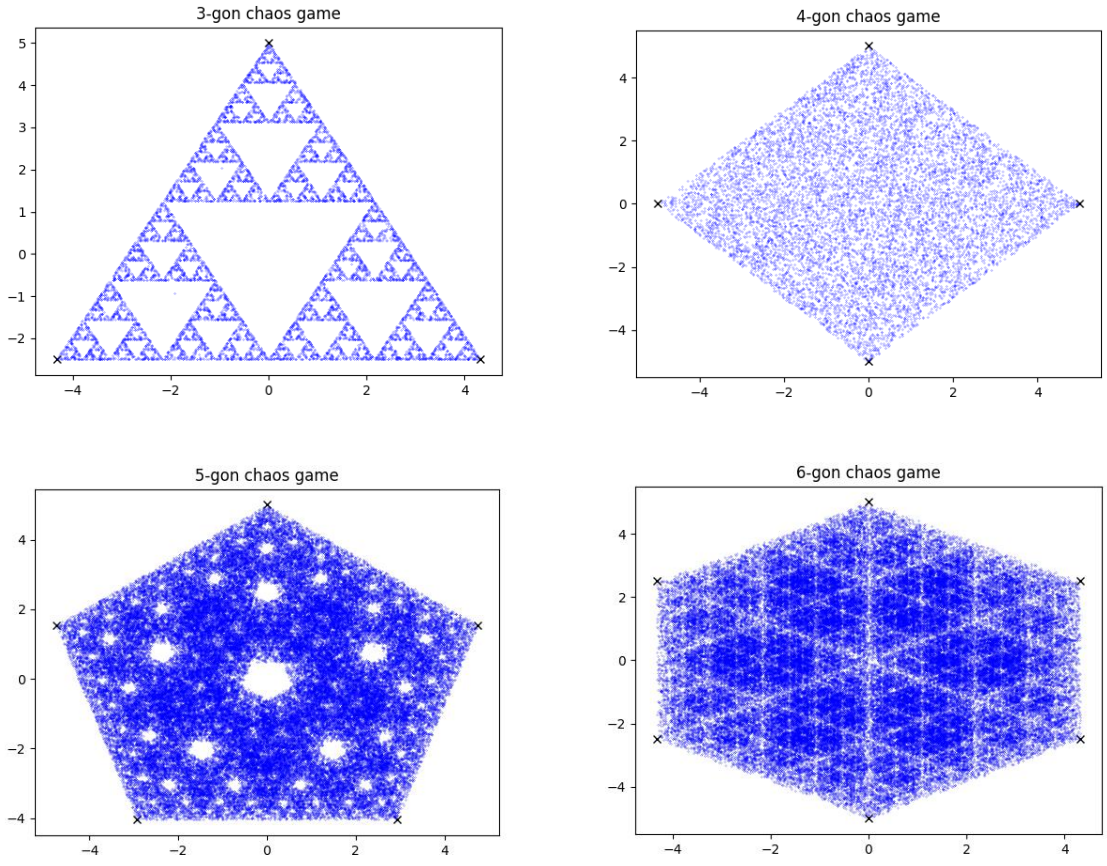
Also, I tried mucking about with some more complicated functions instead of the logistic map.



I still want to speed up my code, could use numba.njit in my python script or rewrite in different language. I think for now I will wait and see how I feel in a few days. Kinda fractaled out right now :)

Chaos Game

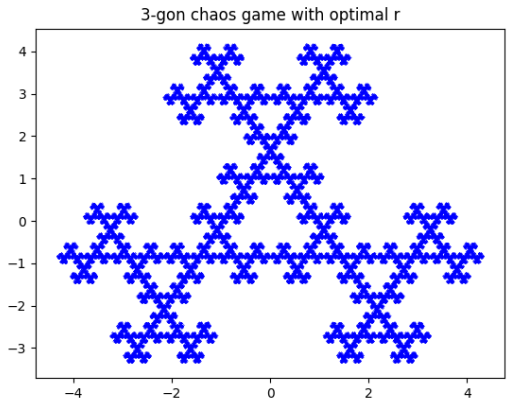
The Sirpinksi triangle through chaos game is (I think) the first fractal I ever coded way back. The chaos game works by starting with a random point and picking a random vertex and then moving to halfway between the current position and the chosen vertex. Then repeat. It is really simple to code as quick to render.

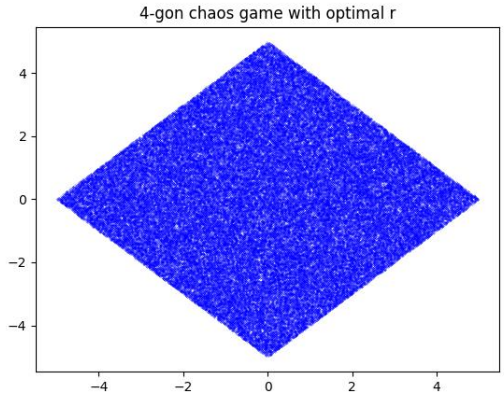


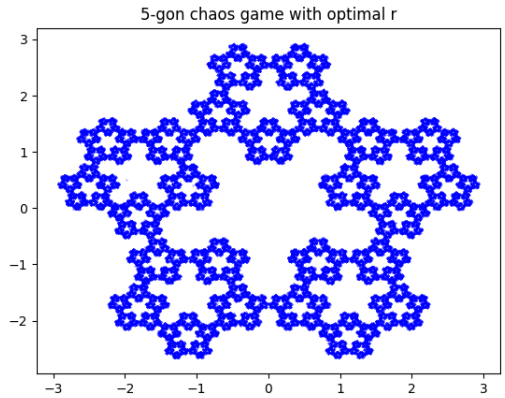
Looking at the chaos game applied to 3-6 gon's there is clear fractal behaviour in all except the square. However, I read on https://en.wikipedia.org/wiki/Chaos_game we change this by adjusting the ratio of distance which we go towards a chosen vertex. And it turns out there is an optimal ratio to see self-similar fractals generated by the chaos game in an N-gon (N > 4). Im not going to pretend I know how this formula for this ratio was calculated, but this is it:

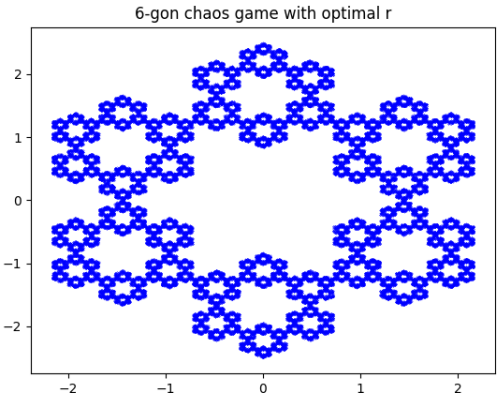
$$r = \frac{1 + 2a}{2 + 2a}, \qquad \text{where } a = \sum_{x=1}^{\lfloor N/4 \rfloor} x(\pi - \theta)$$

And θ is the internal angle of a N-gon.

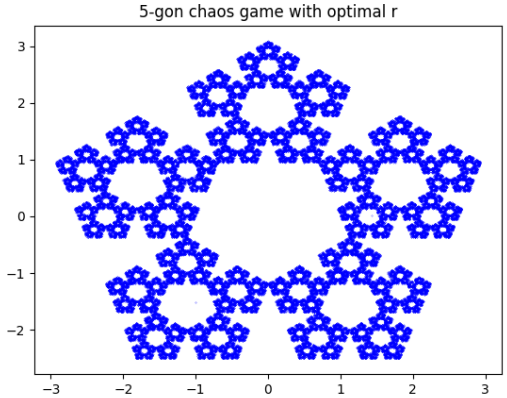


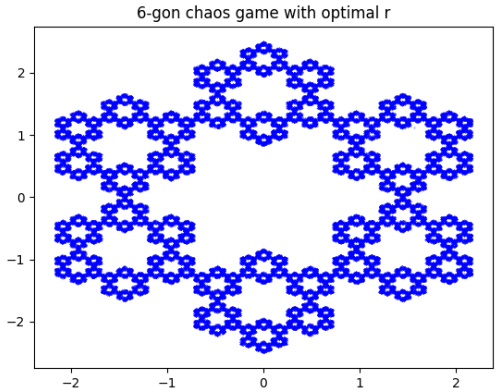






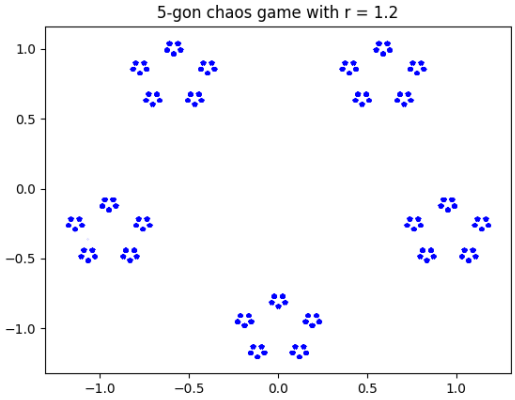
You can see the self-similarity in the 5 and 6-gon's. Sqaure still doing nothing. And what the hell happened to the triangle? Turns out I had a bug in my code so that the next point was calculated as (v – c) * r, where v is the chosen vertex, c is the current point and r is the ratio. Rather then (v + c) * r. Fixed this and square still does nothing.

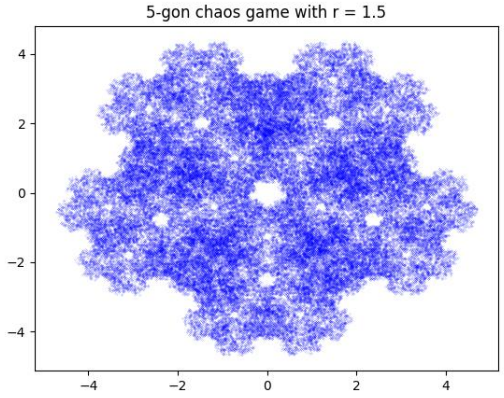


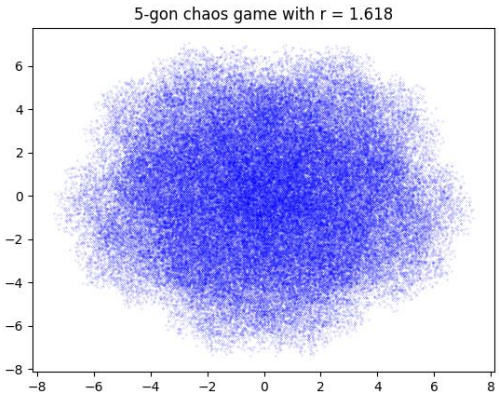


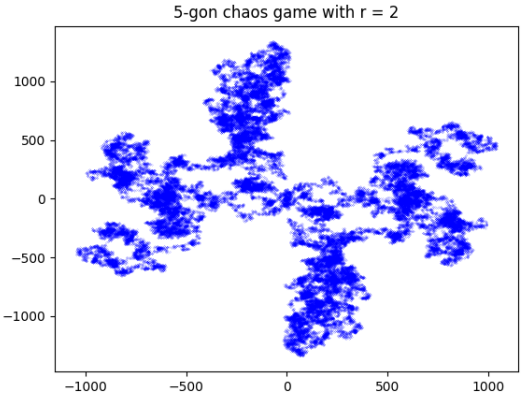
Turns out this bug only affects the plot for odd values of N.

Again from the wiki page, we can set actually r <= 2 without the points diverging.

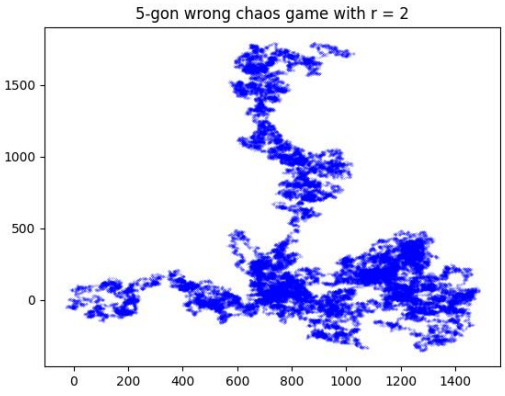


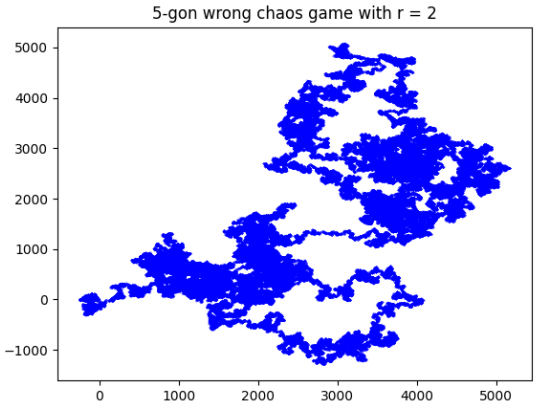






This gives rise to a bunch of unexpected results (at least to me) and at r = 2 is the most interesting plot, there is clear rotational symmetry (about 180°) in the plot and every new run results in a completely different pattern, but the rotational symmetry is maintained. After this I tried used r = 2, with the “wrong” method I used above.

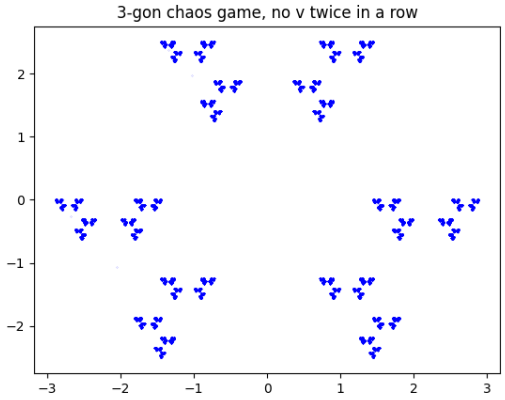


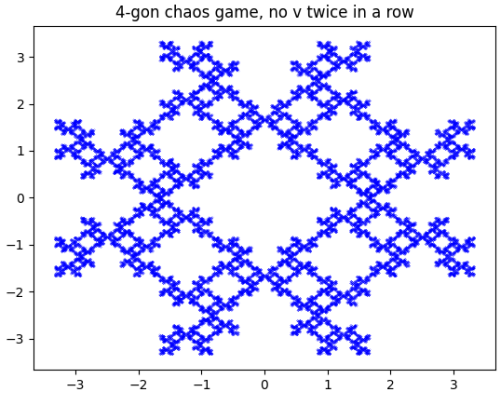


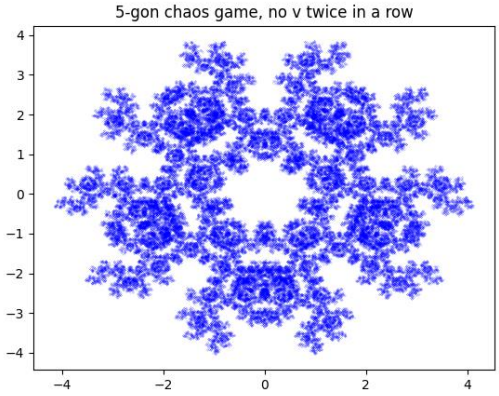
This made some asymmetric but still bounded plots. The right plot above shows 10^7 iterations, and the game did not result in a diverging point, even though the values went way beyond the initial circle of radius 1.

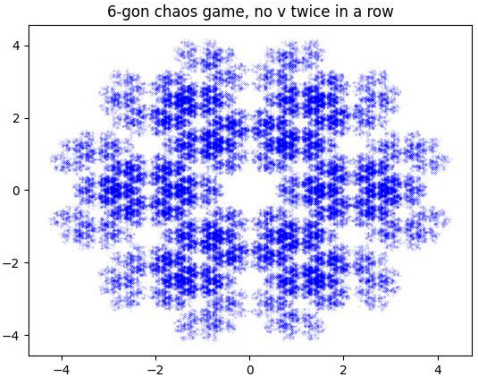
Further note, I suspect that it is almost impossible to tell what N is when r = 2.

Another thing we can do is change the rules of the game, for example make it so a vertex cannot be chosen twice in a row. Going back to r = 0.5 (basic chaos game ratio) and applying this rule we get:

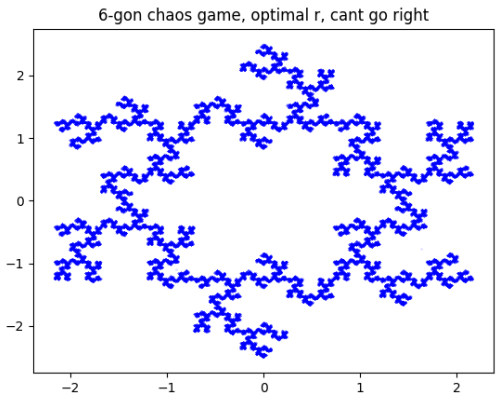


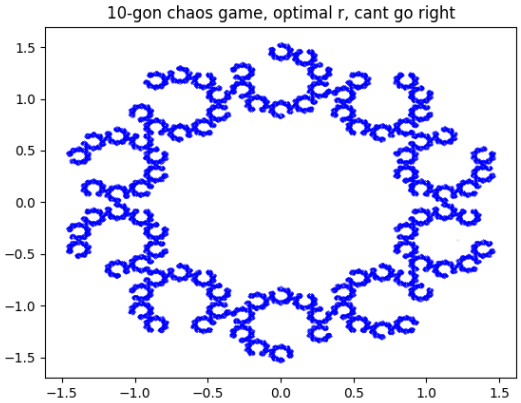




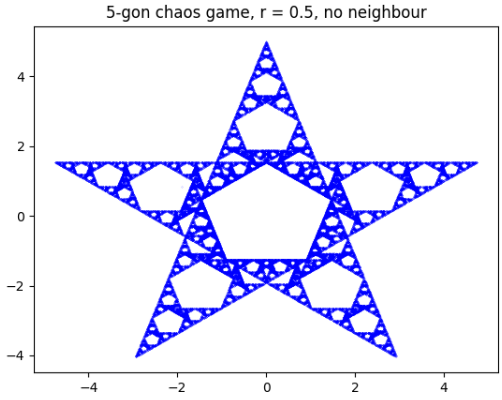


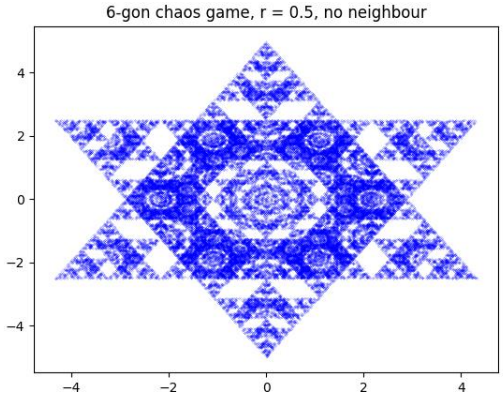
Finally, some interesting behaviour from the square. Now I am going to experiment and combine these rules and maybe come up with some new ones.



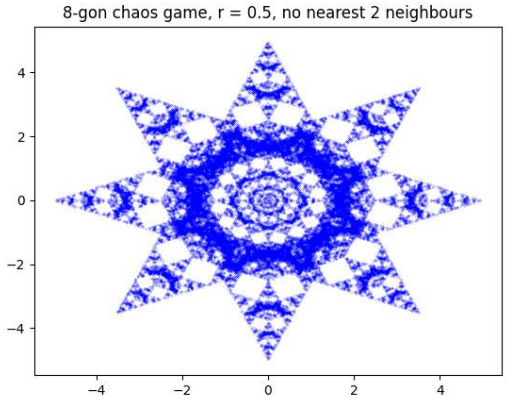


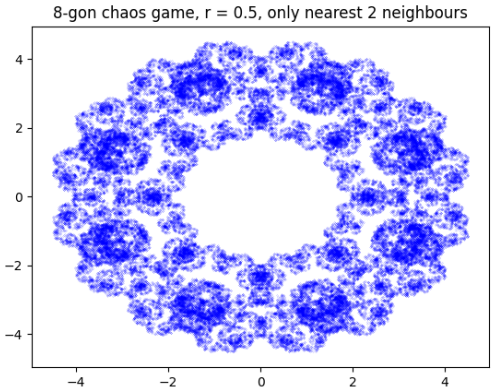
For these two, the next vertex cannot be directly right (anticlockwise) of the current vertex. As N increases it (unsurprisingly) approaches the standard optimal ratio pattern. 3 and 4-gon are in the chaosgame folder on github.



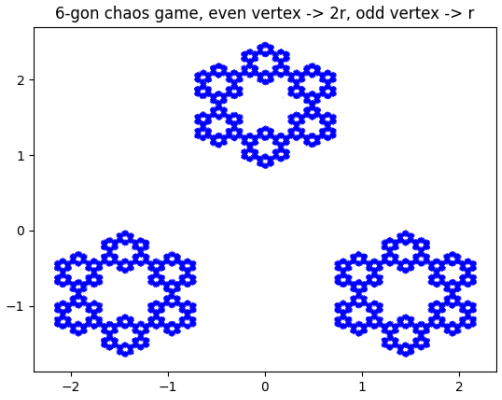


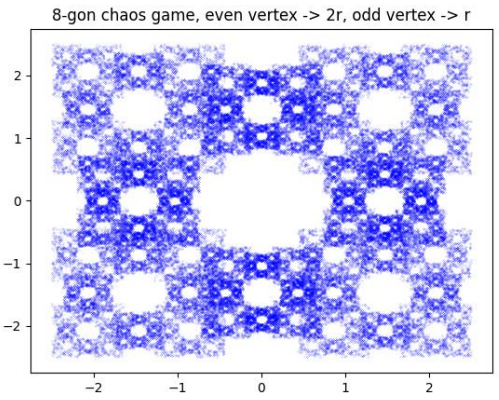
These two are with r = 0.5 and cannot to a direct neighbour of the current vertex next.





These two are kind of opposites (in their method of construction, no idea if this claim is anything close to true) in that in the left plot, only a vertex not within 2 of the current can be selected and in the right only the 2 nearest neighbours can be selected.





These two use the optimal ratio if the vertex is odd, and 2r if the vertex is even. Was very surprised with the difference between the 6 and 8-gon. 7 and 10-gon examples on git. Maybe ill do more someday these are fun to play around with and you will constantly find yourself surprised at what comes out.