3.6 experiment: Orbits, Explosions, and Zeroes

For this project, I got to play around with three different functions and study how their orbits behave under iteration. I wrote a Python script (with help from ChatGPT) to automate the whole process, using libraries like `matplotlib`, `seaborn`, `numpy`, and `pandas`. I was already somewhat familiar with these tools, but this project helped me get much more comfortable using them together —especially when things broke and I had to fix them!

Function A: $f(x) = x^2 - 2$

This function is a classic example in chaos theory, and it showed some really interesting patterns. No matter what seed I chose between -2 and 2, the orbit usually settled into a range between about 1.5 and 2.0 in absolute value. I saw values around 1.7 pop up a lot. What caught my attention was how things started out in a predictable way—many orbits began near 1.999—but pretty quickly, the values started flipping signs and bouncing around in a way that seemed random. It was a great example of how chaos can arise from a really simple rule.

Function B: $f(x) = x^2 - 10$

This one was wild. I wanted to see what would happen if I used a more "aggressive" function, and this definitely delivered. Even starting with small seeds like 0.001, the values exploded in just a few steps. I ran the program on 10 different seeds, and in every case, the orbit overflowed by the 9th iteration—that is, the numbers got so big that Python couldn't even store them. I had to go back and tweak my code to catch this and avoid crashes. That little debugging detour taught me a lot about how to handle numerical instability in iterative processes.

Function C: The Doubling Function

The doubling function was my favorite in terms of how clean and mathematical it felt. It's defined as:

Every time you apply it, you're basically shifting the binary digits of x to the left. I didn't know that at first, but once I realized it, everything made a lot more sense. I noticed that no matter what seed I picked (as long as it was in $[0, 1)$), the orbit eventually reached 0. It usually took around 50 steps or so, but the number of iterations depended on how "deep" the binary expansion was.

To test this, I tried seeds like 0.00000000011111111111111—long decimal numbers with lots of tiny digits. And yes, it did take more iterations to get to zero, but even those didn't make it past 100 steps. I also played around with known rational values like $\frac{1}{5}$ and $\frac{1}{9}$, which are supposed to have periodic orbits. For a few iterations, they actually did follow their expected cycles! But since computers represent numbers approximately, the orbits eventually got distorted and the values collapsed to zero, just like all the others. It was kind of mind-blowing to see that even theoretically neat numbers eventually "crash" in floating point.

Final Thoughts

Each of these functions taught me something different. Function A introduced me to chaos, Function B showed me how fast things can blow up, and Function C gave me a taste of symbolic dynamics and binary behavior. Along the way, I also leveled up my Python skills—handling overflow errors, reshaping data, and exporting results as PDF tables. I started this just wanting to understand some math, but I ended up learning about how computers process numbers too. Not a bad side quest!