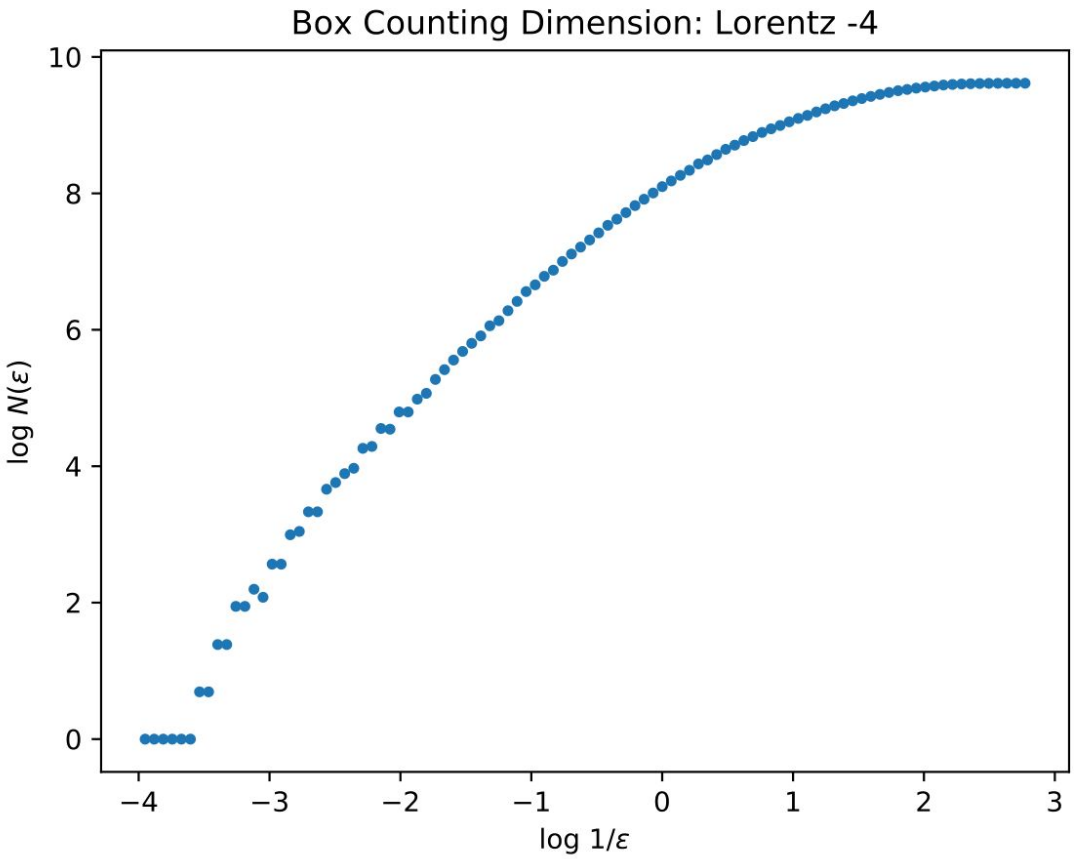
1.

To accomplish this algorithm, I loop through each point in the produced Lorentz trajectory, and increment the corresponding epsilon-sized bin in a numpy histogram. The number of nonzero bins becomes my N(ε). All that's left is to loop over a set of ε's to produce the plots seen below.

2a.

With the trajectory produced from hw9.1a:

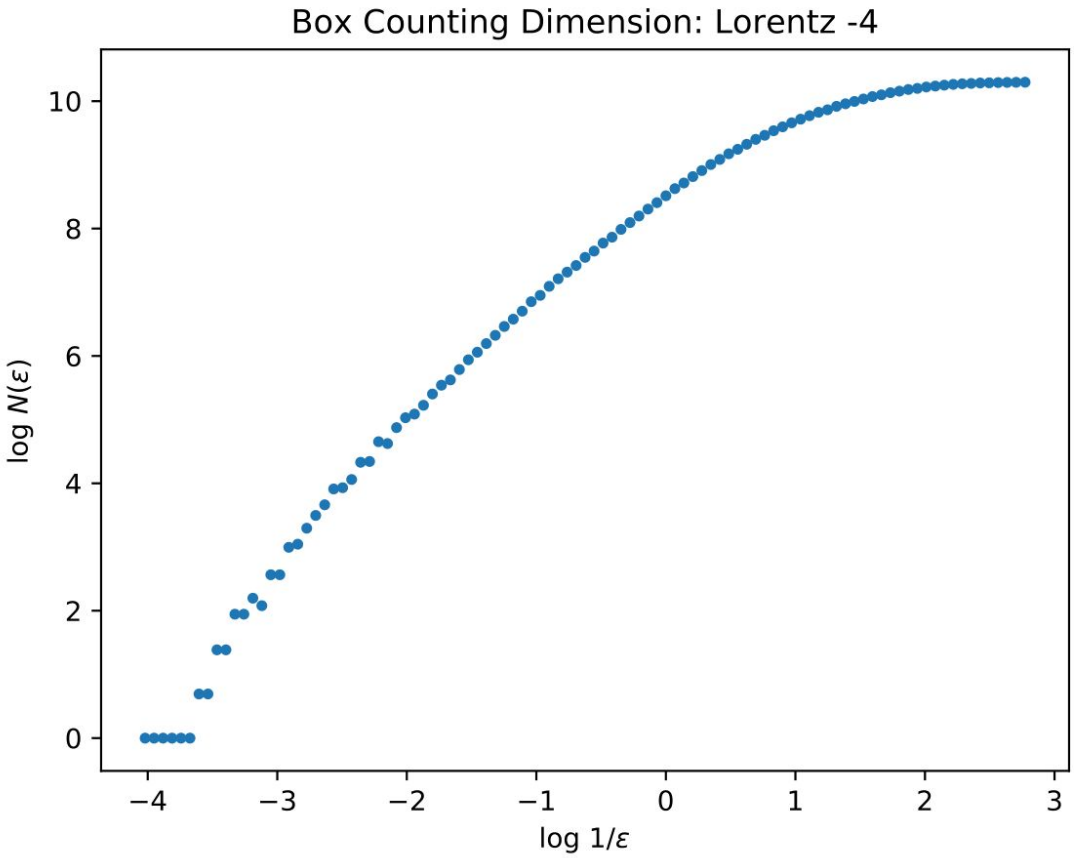


I was able to go down to ε = 0.0625 before my computer ran out of program memory and crashed the program.

The above plot shows the more-or-less linear region where we would extract the capacity dimension (maybe -3 < log(1/ε) < 0), as well as the beginning of the level region at small ε (large log(1/ε) > 2), indicating that the ε balls are sufficiently small for the given trajectory (meaning most bins that have a nonzero value have a value of 1). This is what I would expect for the most part. The zeroes at the bottom left indicate where the ε ball becomes too large, and encapsulates all the points in the trajectory.

2b.

Doubling the trajectory duration, the results look like:



This is very similar to (2a) in shape. The values are a bit different, which is to be expected, since we are adding twice the amount of points, but this shows consistency over different durations of trajectories along the same attractor.

I 100% trust the longer trajectory more than the shorter. If I could, I would want an infinitely long trajectory to calculate fractal dimension, but I can't do that, so I'll take the longest one I can get. The more data the better. Sparse regions in the first trajectory may become more dense in the longer one, causing the capacity dimension to increase (up to the limit, which is what the infinite trajectory would give).

3.

One immediate option is to remove bins once they have a nonzero value (i.e. 1). This would save us the time of checking if the next point is in that bin, since we already have counted that bin towards N(ε). Then we could also change the bins to only hold a boolean instead of an int, to save size as well.