CTA200 2022 Assignment 3

Patrick Horlaville

Question 1

To perform the required iteration, I first set up my x and y dimensional axes. To start with, they are set 100 in length, each covering uniformly the interval between -2 and 2, which means that my complex grid encloses 100x100=10,000 points.

The `iterate()` function allows to perform the indicated iteration over a user-specified number of iteration steps starting from complex number $z_0 = 0$ and using any complex point c. If at any step, the value of $|z|$ goes to infinity, the iteration stops and the final value for the norm of z is set to be infinity. The number of steps that was used to reach this point is retrievable along with $|z|$. If the iteration reaches the total number of iteration steps and $|z|$ is not infinity, $|z|$ is retrieved along with `None` as "the number of steps before divergence" to indicate that $|z|$ has not diverged.

The function `iterate()` can be applied on each point on the grid. It take roughly 1ms to run for one c value for 100 iterations, hence running 10,000 c points (all our grid) over 100 iterations takes about 10 seconds to complete. The resulting iterated values and number of steps are stored. The iterated values are sent through the `booling()` function, which turns any non-infinity entry into `True` and any infinity entry into `False`.

That way, each point on the grid is attributed a `True` or `False` value depending on whether or not the corresponding c to this point on the grid has yielded a divergent $|z|$ during the iteration process.

The `matplotlib.pyplot.contourf` function is then used, along with a binary color map, to represent the distribution of those `True` and `False` values on the grid. I have to admit I am not quite satisfied with this method and I wish I had found a truly binary color mapping tool in python. The few searches and lots of tests I have conducted were not conclusive. Nevertheless it accurately depicts what I aimed to plot:

From the plot, it seems like any complex point on the grid whose imaginary component $\lessapprox 0.25$ does not diverge when being iterated over with `iterate()`.

Then, we can make use of "the number of steps before divergence" for the second plot. A color map, still using `matplotlib.pyplot.contourf`, is used. For each point, we have either a `None` value or an integer value depending on whether $|z|$ diverged during the iteration or not. From our first plot, it seems like all points below $y \approx 0.25$ are convergent, so we limit our plot to $y \in [0, 2]$ to have a better insight on the features of the part where points are divergent. This gives us figure 2:

Question 2

First, the equations are set up pretty easily with a defined `eqns()` function, which deals with 3 ODEs for each of our variable. The values of the parameters and initial conditions are then set with W_0 and srb. We set a time scale of integration from 0 to 60 divided in 6000 time steps, so as to have a time step of 0.01.

The function `odeint()` is then used to integrate our system of ODEs with the specified W_0, srb values and time domain.

From the output of `odeint()`, we can pick out the evolution of our system in each spatial dimension. In order to replicate Figure 1 from Lorenz, we first pick out the Y dimension and look at its evolution through the first 3000 time steps, plotting 3 times 1000 steps. We have the following figures:

To reproduce Figure 2, we pick out the solution of our equations between time steps 1400 and 1900. We can then plot the Y against the Z solution and the Y against the X solution, which corresponds to the plots of Figure 2 from Lorenz:

We can repeat the solving of the ODEs system with a slightly different set of initial conditions. First, we define that new set W_0' according to the problem statement. To compare the two solutions, analyze dimension by dimension. We take the sum of the squared difference between each X, Y and Z component. Take the square root to retrieve the distance. Look at how that distance evolves in time and we get: