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 some complex point c as prescribed by the problem set. If at any step, the value of $|z|$ goes to infinity, the iteration stops and the final value for the norm of z `norm` is set to be infinity. The number of steps `nsteps` 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 and `nsteps` is set to `None` as 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 `norm`'s and `nsteps`'s 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:

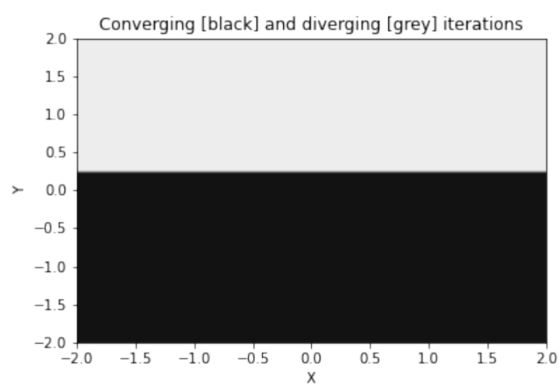

Figure 1: Here the complex grid is shown, where the divergent/convergent behavior of each point (when input into `iterate()`) is shown either in black (for convergent) or in grey (for divergent).

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

Then, we look into `nsteps` 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 the points are divergent. This gives us figure 2:

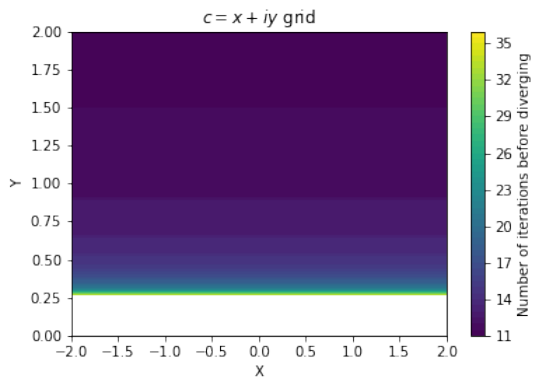

Figure 2: Here the complex grid is shown in the imaginary domain only between 0 and 2. Each point in the colored region reaches infinity after a number of steps that is shown. As we can see, it requires less and less steps before reaching infinity as y increases.

My results show that on the grid, complex numbers with an imaginary component approximately inferior to 0.25, when input into the `iterate()` function as c, do not make $|z|$ diverge to infinity. Rather, this divergence behavior happens when $\Im(c) \gtrsim 0.25$. As $\Im(c)$ increases, the number of steps needed to reach the divergence decreases. This makes sense as greater components for $\Im(c)$ contribute to increase $|z|$ in the iteration, therefore it increases more rapidly towards infinity.

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 [1], 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:

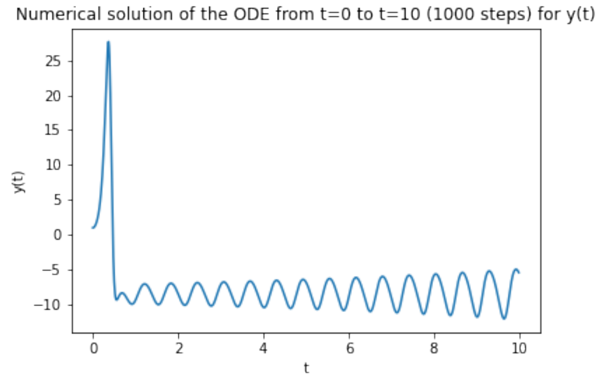

Figure 3: Numerical solution in the Y-dimensional axis between t=0 and t=10. The behavior of the solution looks strongly similar to that of Lorenz's [1].

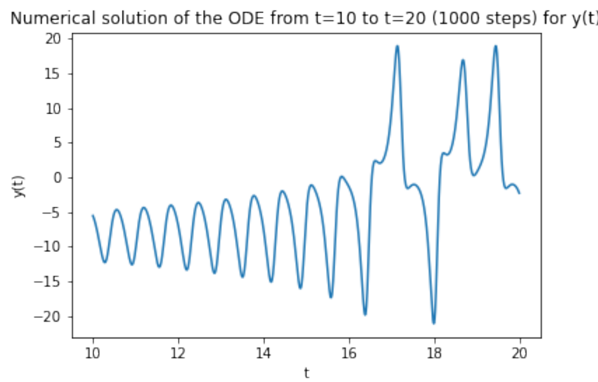

Figure 4: Numerical solution in the Y-dimensional axis between t=10 and t=20. Again, the behavior of the solution looks strongly similar to that of Lorenz's [1].

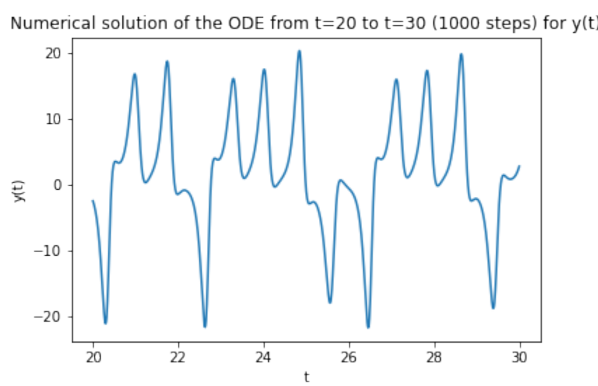

Figure 5: Numerical solution in the Y-dimensional axis between t=20 and t=30. Again, the behavior of the solution looks strongly similar to that of Lorenz's [1].

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:

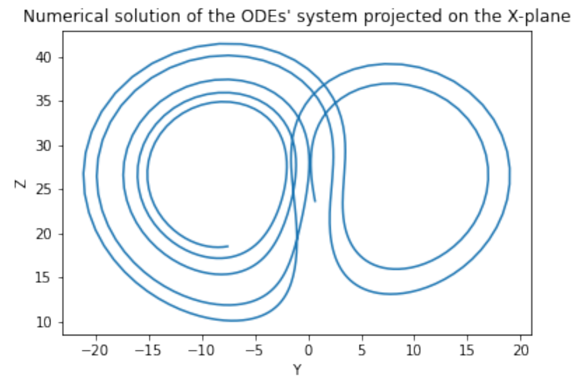

Figure 6: Numerical solution between steps 1400 and 1900 projected on the X-plane. Once more, the behavior of the solution looks strongly similar to that of Lorenz's [1].

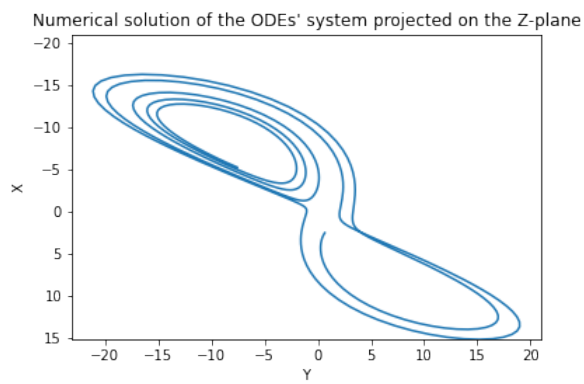

Figure 7: Numerical solution between steps 1400 and 1900 projected on the Z-plane. Again, the behavior of the solution looks strongly similar to that of Lorenz's [1].

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 set. To compare the two solutions, we 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 between each point of W_0 and W_0' at any time t. Look at how that distance evolves in time and we get:

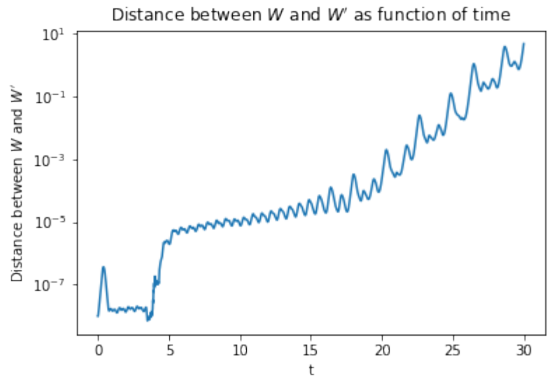

Figure 8: Evolution of the distance between W_0 and W_0' as a function of time, as shown in a semi-log plot. The linearity of the relationship hints towards the exponentially increasing difference with time despite marginally small initial differences between W_0 and W_0'.

Overall I observe a coherent match between my results and those Lorenz's [1]. Small perturbations into the initial conditions of the system of equations reigning our system and the solutions quickly differ; they do so exponentially as a matter of fact as seen in Figure 8. The shapes of the different plots shown are also a good visual match (Figures 3, 4, 5, 6 and 7).

References

[1] Edward N. Lorenz. "Deterministic Nonperiodic Flow". In: *Journal of Atmospheric Sciences* 20.2 (1963). Place: Boston MA, USA Publisher: American Meteorological Society, pp. 130–141. DOI: `10.1175/1520-0469(1963)020<0130:DNF>2.0.CO;2`. URL: `https://journals.ametsoc.org/view/journals/atsc/20/2/1520-0469_1963_020_0130_dnf_2_0_co_2.xml`.