CTA200 2022 Assignment 3

Ian Niebres
Tuesday, May 10 2022

Question 1

In this question, we iterated the equation $z_{n+1} = z_n^2 + c$ for each point $c = x + iy$ in the complex plane for $-2 < x < 2$ and $-2 < y < 2$. A function called `iterate` was written in the file `iteration.py` which performs this iteration `N` times given an initial $z_0 = 0$ and $c = x + iy$. This function returns the values of z, its modulus, and the iteration number (at which it diverges, if it diverges).

Two arrays were made, called `x_vals` and `y_vals`, using `numpy.arange` that contain 400 values from -2 to 2 equally spaced apart. The function `numpy.meshgrid` was used with these arrays to create a mesh grid of `x` and `y` values to be used for plotting. A third array (called `zz`), for the third axis, was created with all elements 0 with the same shape as the `x` and `y` arrays. Using a double `for` loop, the elements of this third array was filled with the last element returned by the iteration value for each `x` and `y`.

The function `numpy.nan_to_num` was used to change all `NaN`s in the array to a value over a threshold number. All values equal or below this threshold number were considered stable, and given the element corresponding to the value's index was changed to 1. All values over this threshold number were considered to be divergent, and given a value of 0. At this point, we have enough to plot using `pyplot.imshow` from `matplotlib`. A greyscale colourmap was used since the data is binary. Figure 1 was created where black shows stability of the iteration with $c = x + iy$. White shows divergence.

The `iterate` function was changed, so that it now tracks the iteration number at which it diverges. This was done by using `numpy.where`, and searching for values at which the modulus was above the threshold. Since the function returns the index, it was trivial to obtain the iteration number at which it diverged. A copy of the `zz` array of 0s was made, and updated with this iteration number at divergence. A new plot, Figure 2, was made which shows the number of iterations it took for z to diverge. Note, 0 means the function does not diverge.

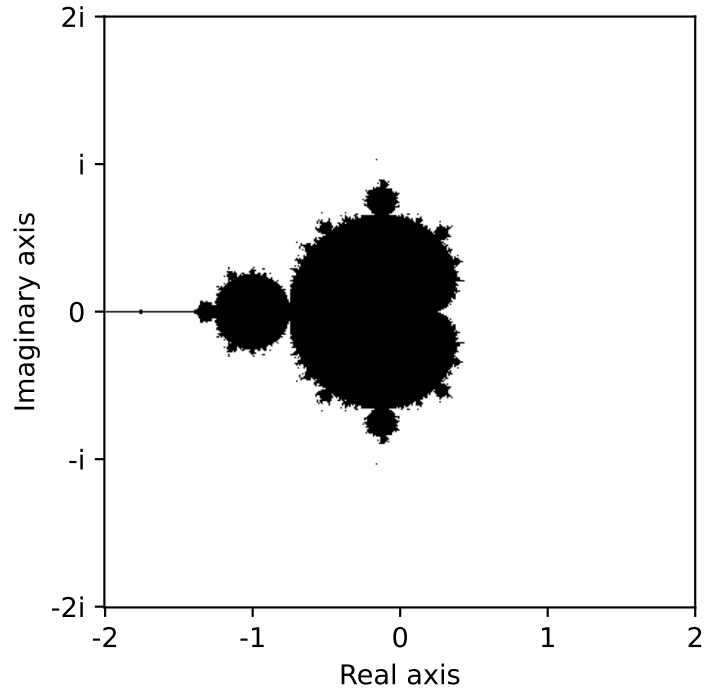

Figure 1: x, y values used in the iteration of $z_{n+1} = z_n^2 + c$ where $c = x + iy$. Black points are bounded, white points diverge.

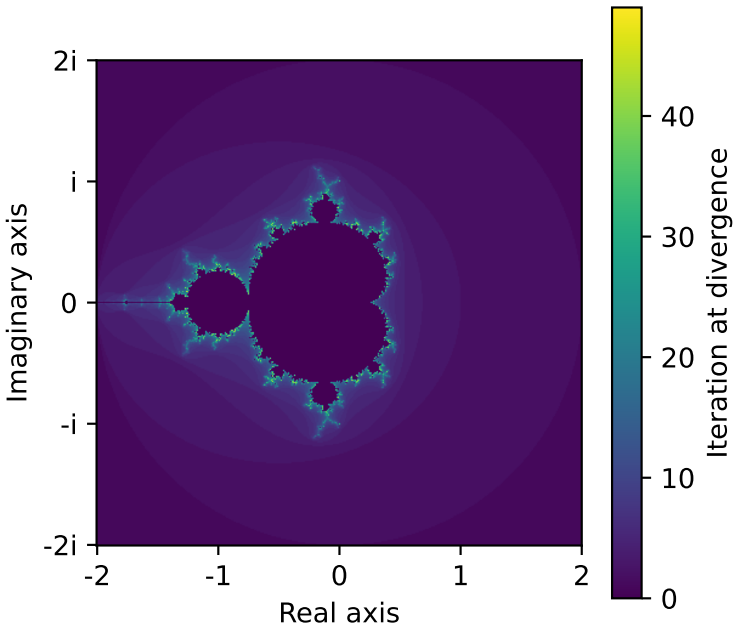

Figure 2: Same as Figure 1, but plot now shows the number of iterations before divergence.

Question 2

In this question, we wish to solve Lorenz' equations,

$$\dot{X} = -\sigma(X + Y) \tag{1}$$
$$\dot{Y} = rX - Y - XZ \tag{2}$$
$$\dot{Z} = -bZ + XY \tag{3}$$

where σ, r, b are the Prandtl number, the Rayleigh number, and a length scale. Note that these are all dimensionless parameters. We will be using `solve_ivp` from the package `scipy.integrate`.

Task 1

Firstly, the constants (parameters) were assigned values `[sigma, r, b] = [10., 28., 8./3.]`. As well, tolerances to be used for the `solve_ivp` function were defined. Lastly, the initial conditions `W0 = [0., 1., 0.]` were defined. A function, `lorenz`, was written to be used in `solve_ivp`. This function is of the right hand side of the ODEs in Equation (1) to Equation (3).

Task 2

Another function, called `solve_lorenz`, was made, which takes in a list of the initial conditions, and returns the solved ODEs using `solve_ivp`, in conjunction with the previously assigned variables and tolerances in Task 1.

Task 3

We were tasked to recreate Figure 1 from Lorenz' original paper. The recreated figure is Figure 3. This plot was made using `pyplot` from `matplotlib`. This figure is a plot of Y values of the numerical solution to the ODEs in Equation (1) to Equation (3).

Task 4

We were then tasked to recreate Figure 2, which are phase space plots in the Y-X and Y-Z plane. Similar to what we did in Task 1, but instead we are looking through iterations 1400 to 1900. See Figure 4 and plotting different values on the axes.

Task 5

Now we wish to perturb the initial conditions a small amount, and look at how the solution changes over time. This was done by perturbing `W0` $\rightarrow$ `W0'` where `W0' = W0 + [0., 1.e-8, 0.]`. New solutions were obtained using `W0'` as initial conditions in the `solve_lorenz` function. The distance between the two solutions `W` and `W'` were calculated using the equation,

$$d = \sqrt{(X - X')^2 + (Y - Y')^2 + (Z - Z')},$$

where (X, Y, Z) and (X', Y', Z') are the solutions to the Lorenz equations with `W0` and `W0'` initial conditions, respectively. The distance was then plotted on a semilog plot using `pyplot.semilogy`, where the time on the x-axis is linear, and the distance on the y-axis is logarithmic. See Figure 5.

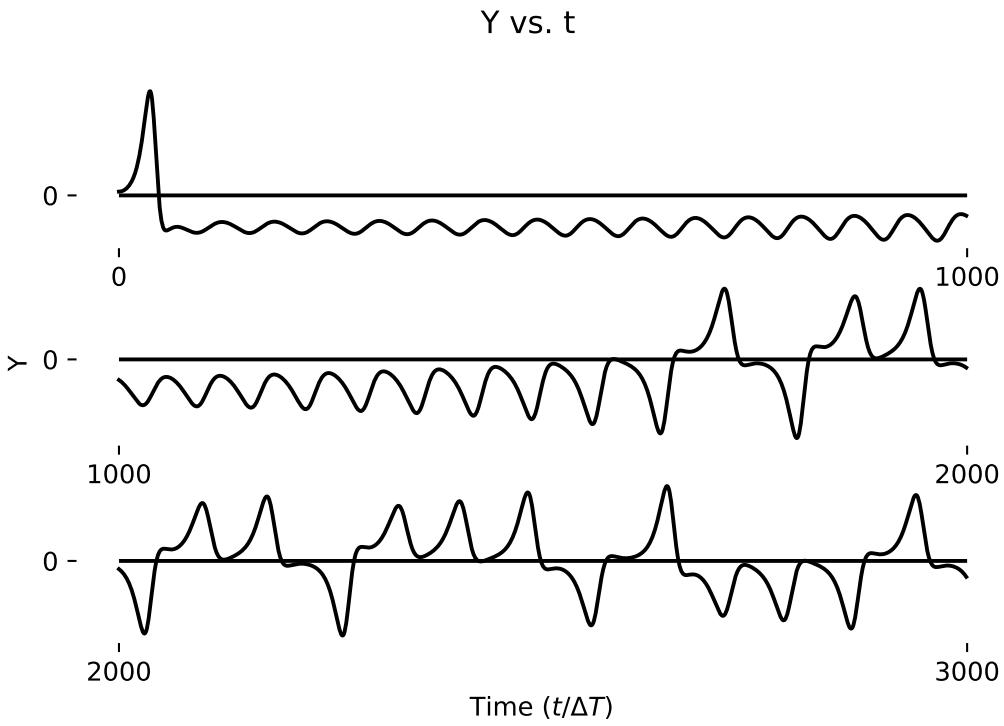

Figure 3: Plots of Y values of the numerical solution to the ODEs in Equation (1) to Equation (3). Note that $t = 60$ and $\Delta T = 0.01$ s. Top: first 1000 iterations. Middle: iteration 1000 to 2000. Bottom: iteration 2000 to 3000.

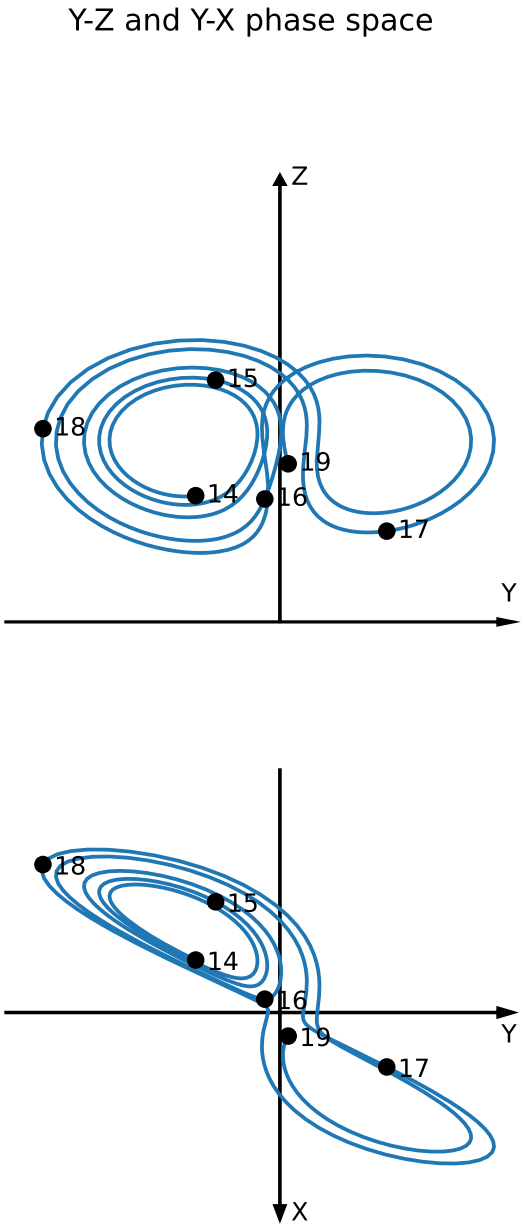

Figure 4: Phase space plots of the numerical solution to the ODEs in Equation (1) to Equation (3). Note, markers labeled 14 to 19 indicate the solution at iterations 1400 to 1900, respectively. Top: Y-Z. 2000. Bottom: Y-X phase space.

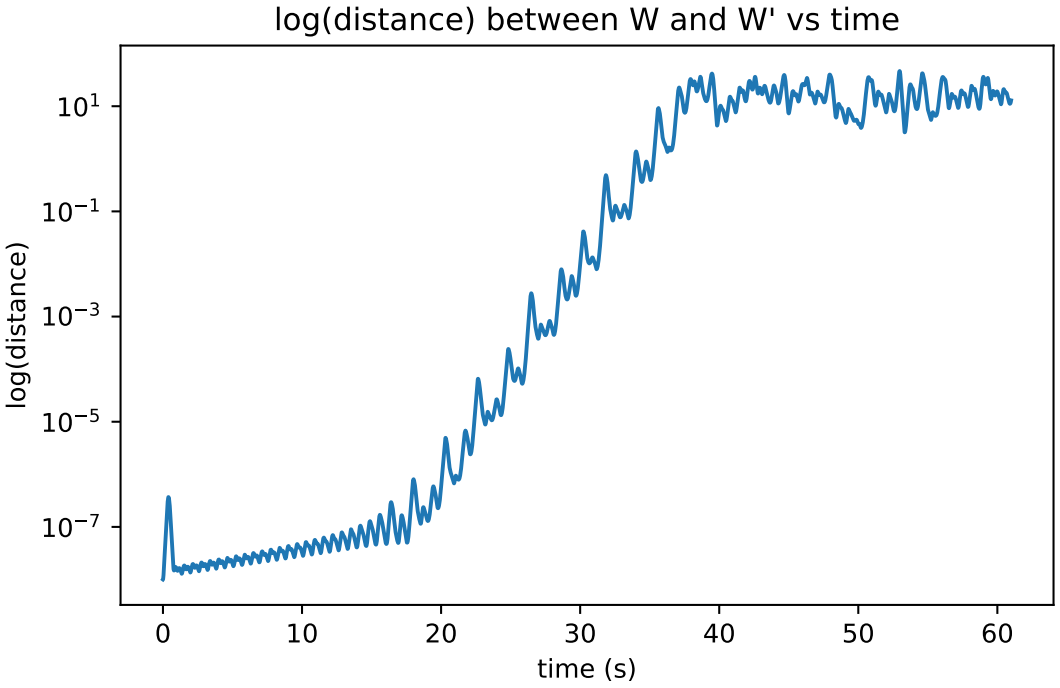

Figure 5: Logarithm of the distance between solutions `W` and `W'` with respective initial conditions `W0` and `W0'` as a function of time.