Assignment 2

Leandro Rizk
leo.rizk@mail.utoronto.ca
CTA200 – University of Toronto

8 May 2021

1 Question 1

1.1 Methods

I wrote two functions: deriv1 uses method 1 and deriv2 uses method 2. Each function takes a callable (i.e. another function), a value for x_0, and a step size (h). I used each function to calculate the derivative of $sin(x)$ at $x = 0.1$ for an array of 99 step sizes ranging from 0.01 to 0.99. The derivative of $sin(x)$ is simply $cos(x)$; so to get the relative error, I performed the following operation:

$$\text{relative error} = \left| \frac{\text{numerical approximation} - \cos(0.1)}{\cos(0.1)} \right|,$$

where the numerical approximation is the array resulting from calling deriv1 or deriv2. Plotting both methods' relative error arrays vs. the step size array in log scale gives **Figure 1**.

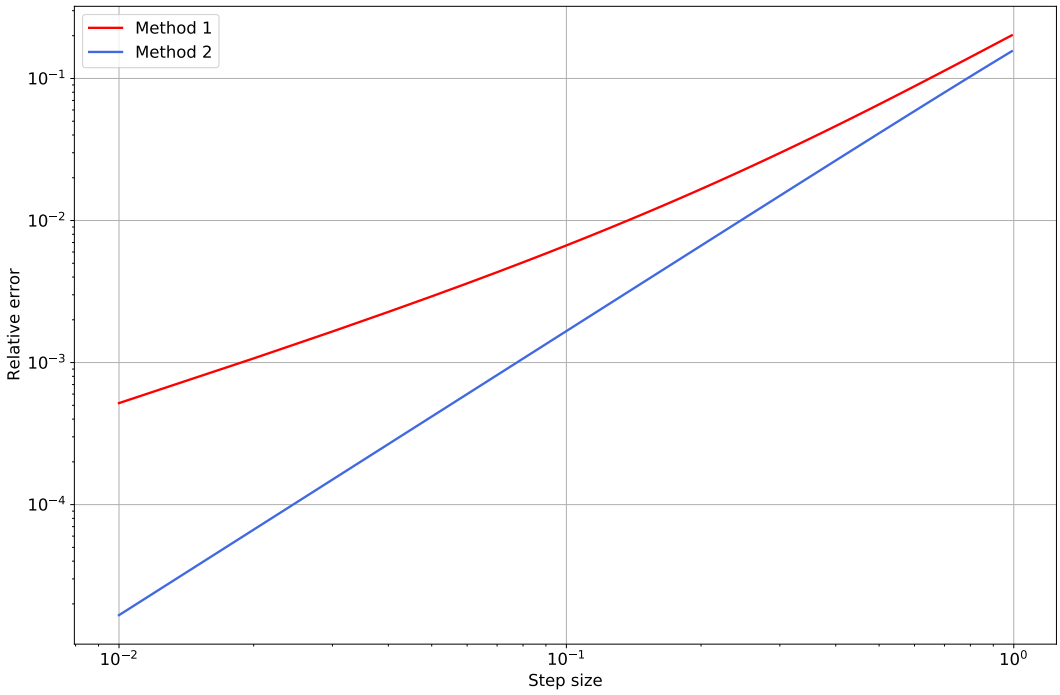

Figure 1: Relative error of numerically evaluating the derivative of $sin(x)$ at $x = 0.1$ in relation to step size

1.2 Analysis

The relation between the relative error and the step size is roughly linear in logarithmic scales. This is especially true for method 2. A constant coefficient in log scale corresponds to a constant exponent in linear scale. In a sense, the slope in log scale represents the order of magnitude by which the step size affects the relative error. Since the slope in log scale for method 2 is about 1.99 (≈ 2), the relation between relative error and step size for this method should be quadratic. As seen in **Figure 1**, method 2 produces a consistently smaller error than method 1 over the range of chosen step sizes.

2 Question 2

2.1 Methods

I created two arrays for x and y, each containing 299 evenly spaced elements from -2 to 2 (exclusive). I then created a Python dictionary, with the key being every possible complex number c obtained from combinations of x and y elements and the values being a list of the 1000 first iterations of z (starting with 0) corresponding to that complex number c. To obtain **Figure 2**, I created an image of the complex plane represented by a 2-D numpy array of the shape (299, 299), initially making all values zero. I wrote a nested for loop that examined the dictionary for every complex number key represented in the 2-D array and that converted the image pixel value from zero to one if np.inf was found in the list of iterations of z for that complex number. **Figure 3** was created in a similar fashion except, instead of converting the image pixel value from zero to one, the pixel value was converted to the index of the first occurrence of np.inf inside the list of iterations of z. If the pixel value remained zero, it was understood that z did not diverge for that point in the complex plane. The resulting colour for zero in the plot was a distinct enough black that it isn't easily confused with a colour corresponding to rapid divergence (represented by purple).

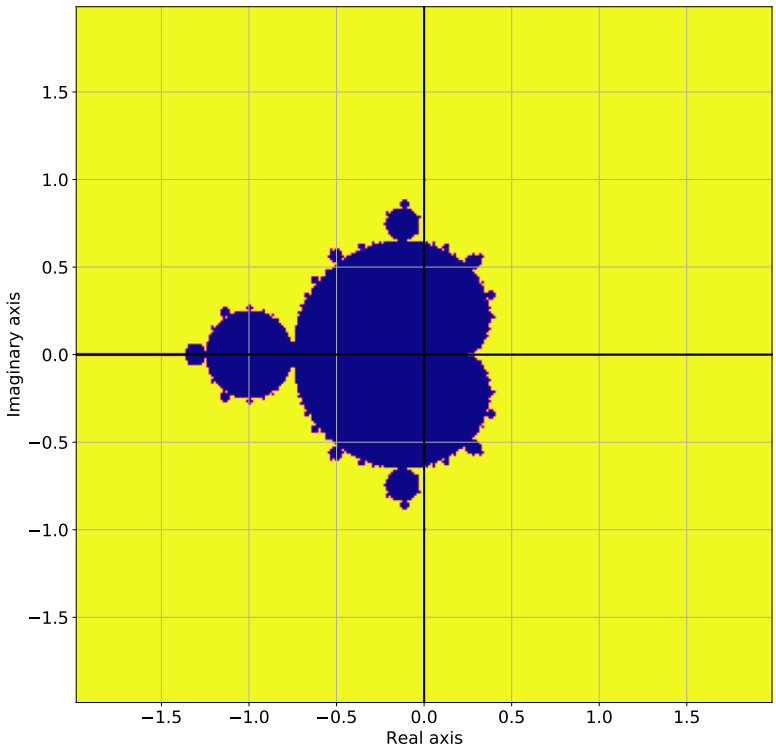

Figure 2: Locations in the complex plane where z diverges (shown in yellow)

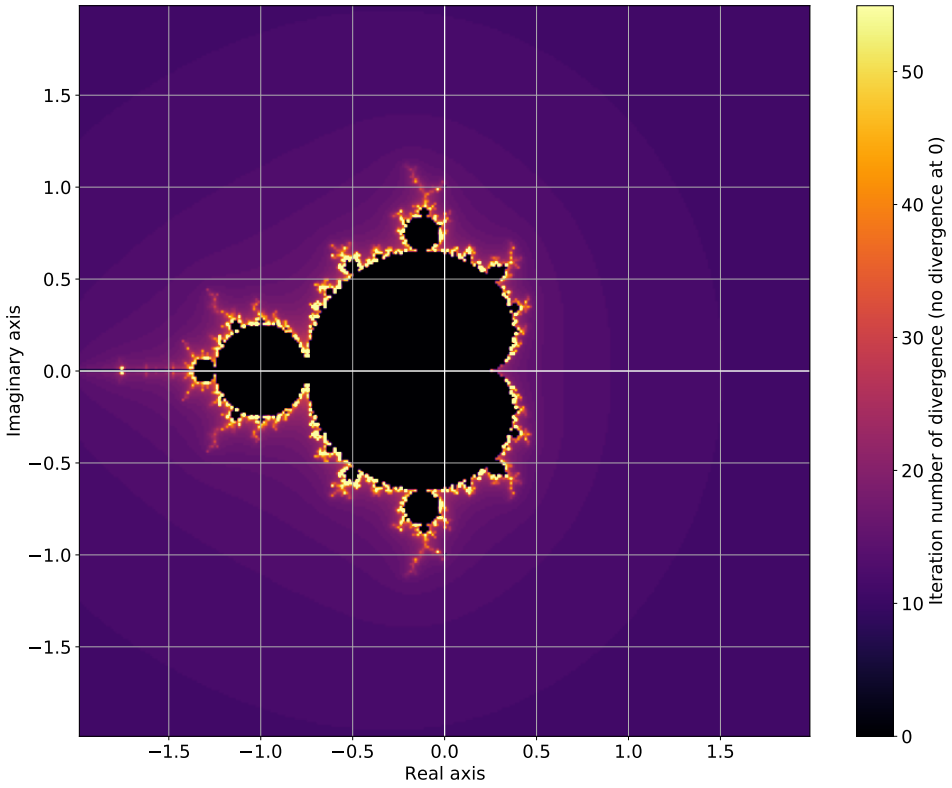

Figure 3: Divergence of z in the complex plane

2.2 Analysis

Figures 2 and **3** show a Mandelbrot fractal. The edges of the fractal have a z that diverges much more slowly than points farther from the fractal. The plot is symmetric along the real axis, so a complex number will have the same behaviour in z as its complex conjugate.

3 Question 3

3.1 Methods

I wrote a SIRmodel function that returns $\frac{dS}{dt}$, $\frac{dI}{dt}$, and $\frac{dR}{dt}$ (as vector) for a given vector $[S, I, R]$. This function is meant to be passed to the function ode from the module scipy.integrate along with initial conditions (that I called SIR_0) and time elements: initial time (t_0), endtime (t_{end}), and timesteps (dt). I set SIR_0 = [999, 1, 0] (thereby setting $N = 1000$), $t_0 = 0$, $t_{end} = 200$, and $dt = 0.1$ and did not later modify these. The parameters of the SIR model β and γ were free to be reassigned before using ode. To simplify using ode, I also wrote a function solveode which streamlines the process: it takes the callable (SIRmodel), the initial conditions vector, and the time arguments, performs the necessary steps for using the function ode (with integrator dopri5), and returns the solution ($S(t)$, $I(t)$, $R(t)$) as a 2-D numpy array as well as the time array as a 1-D numpy array. This allowed me to reassign β and γ and easily solve the differential equations again. In order to make physical sense, γ could not exceed 1.

In adding D, I assumed deaths were only a result of the disease (susceptible and recovered individuals could not die). So this meant that D was only populated from I:

$$\frac{dD}{dt} = \frac{\delta I}{N},$$

for a death rate $\delta \geq 0$. This means that a term $-\frac{\delta I}{N}$ must be added to the definition of $\frac{dI}{dt}$. I repeated the same steps as above, defining the new SIRDmodel callable and passing it to solveode with a new four-element initial conditions vector ($SIRD_0 = [999, 1, 0, 0]$). In order to make physical sense, $\gamma + \delta$ could not exceed 1.

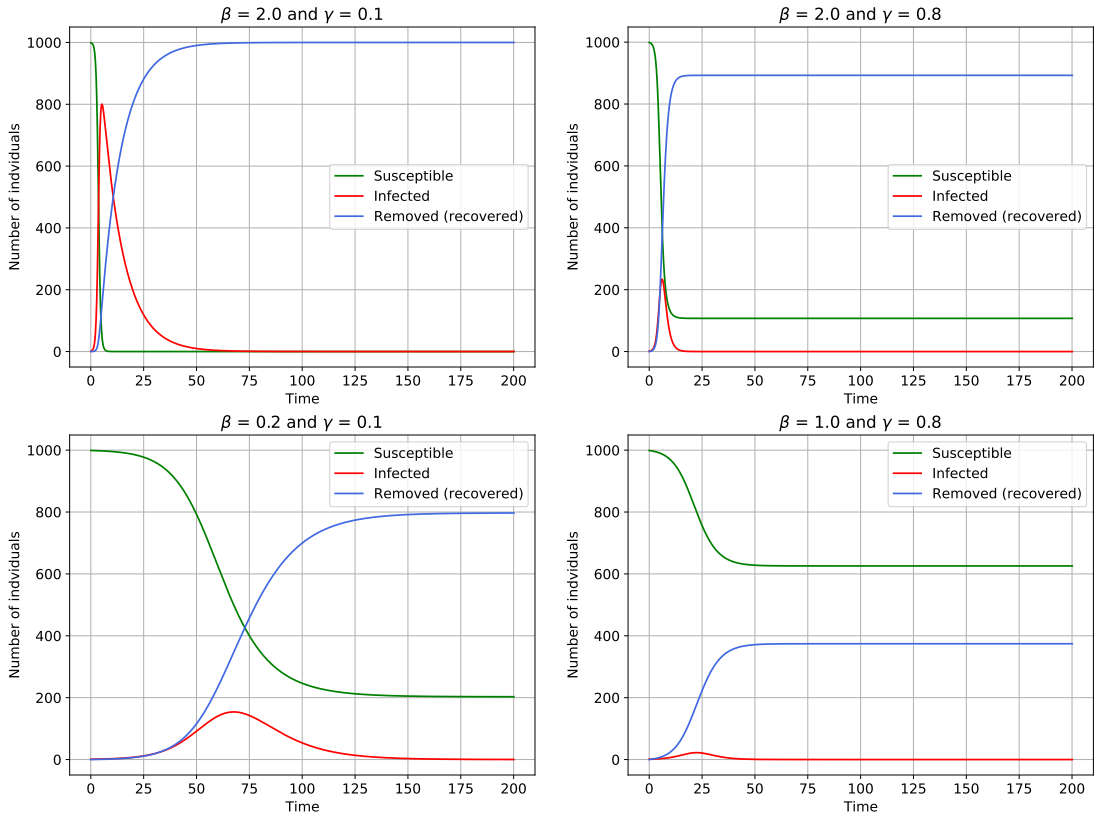

Figure 4: Disease evolution in a population of 1000 individuals (SIR model)

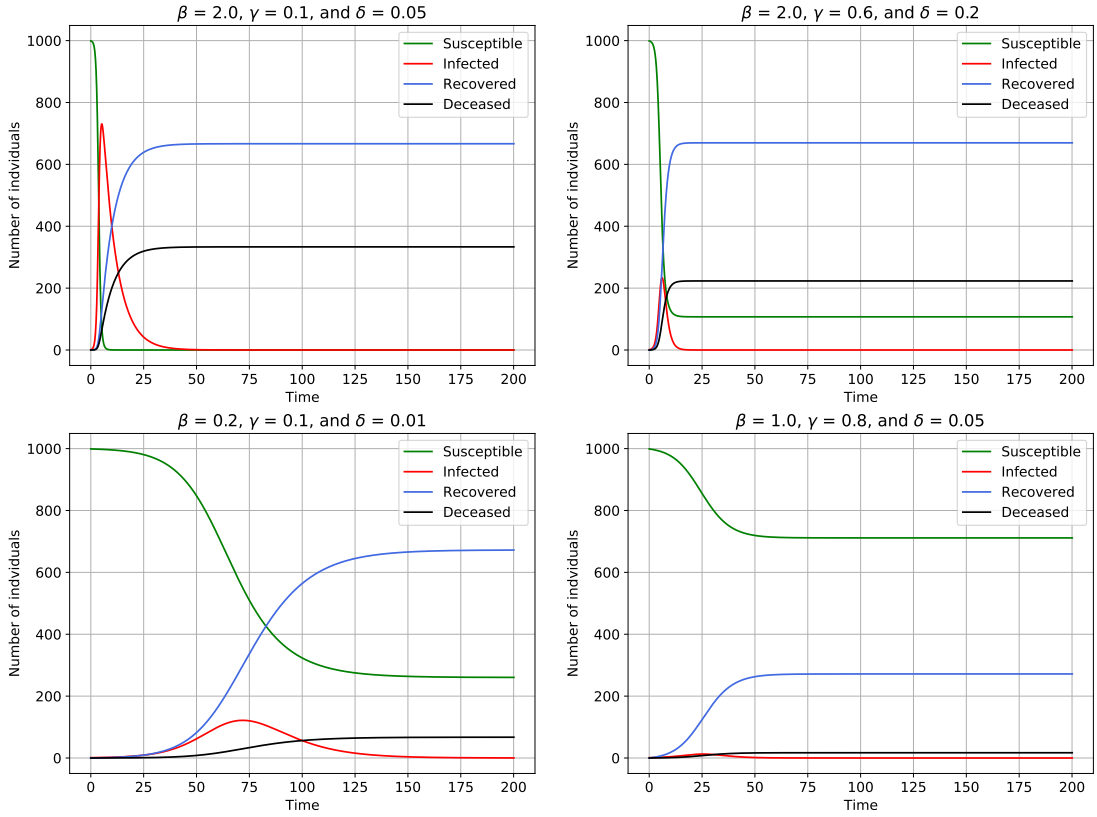

Figure 5: Disease evolution in a population of 1000 individuals (SIRD model)

3.2 Analysis

In **Figure 4**, a high infection rate and a slow recovery results ($\beta = 2.0$, $\gamma = 0.1$) in a very quick conversion of the entire susceptible population to infected and a more gradual conversion from infected to recovered. The same high infection rate but with a much quicker recovery rate ($\beta = 2.0$, $\gamma = 0.8$) quickly converts susceptible to infected and infected to recovered. Since infected individuals recover very quickly, this number cannot build up very much and, as a result, a considerable proportion of the population remains uninfected after the end of the course of the disease. A low infection rate and a slow recovery (($\beta = 0.2$, $\gamma = 0.1$), produces a similar result in terms of disease evolution, but on a much longer timescale. Finally, a low infection rate and a rapid recovery ($\beta = 1.0$, $\gamma = 0.8$) produces very little active infections from susceptible individuals. Relatively few of the population contract the disease by the end of its course.

In **Figure 5**, adding the possibility of death simply splits the "removed" individuals into two categories: recovered and deceased. Of note, the death rate (δ) must always be considered in relation to the recovery rate (γ) and vice versa. For example, in the case of a very small value of δ, a γ of zero still means that every infected individual will eventually die of the disease (100% mortality). For any time t, the sum of susceptible, infected, and removed individuals is always equal to N (1000).