James Elijah Braganza Morillo U1740375H

# Outline

The code begins with introducing a class for solving the ODE using RK4, which creates a smoother and neater way of forming the code structure. The first class of functions we have will be LorenzAttractorRK4 and next we will have another class for bifurcation, logistic map and Lyapunov map functions. This is followed by creating the whole overall architecture of the code by defining the parameters and the initial conditions. We keep the initial values constant (X_0, Y_0, Z_0 = 0.2, 0.2, 0.2) and then further study it in a randomized values.

$\dot{x} = \sigma(y - x)$
$\dot{y} = rx - y - xz$
$\dot{z} = xy - bz$

For a start, we state that the Lorenz equation has two stable points (represented as C and C*) below:

$C \sim (\sqrt{\beta(r-1)}, \sqrt{\beta(r-1)}, (r-1))$
$C^* \sim (-\sqrt{\beta(r-1)}, -\sqrt{\beta(r-1)}, (r-1))$

The class of *LorenzAttractorRK4* will comprise the whole bulk of RK4 execution, the Lorenz Attractor equations, a 3-dimensional plotter 3D Axes imported from mpl_toolkits.mplot3d and a 2-dimensional plotter.

# Sketches

To get snippets of the big picture of the Lorenz attractor, we sketch some of the dependencies. This helps us understand at a glance where the various parts of how I derived the designs for the code I used.

$\dot{x} = \sigma(y - x)$
$\dot{y} = rx - y - xz$
$\dot{z} = xy - bz$

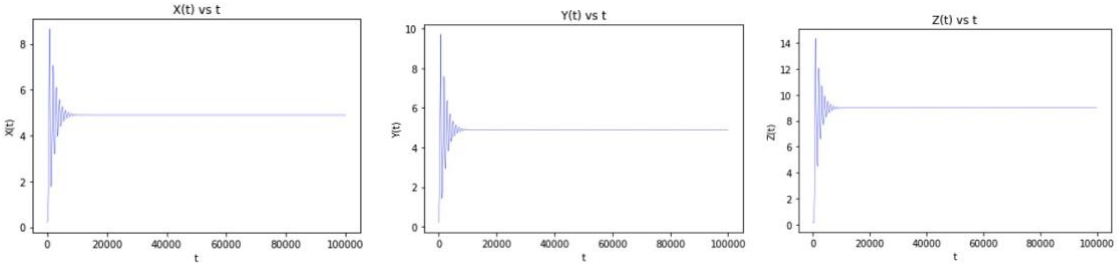
Let us plot X(t)-t, Y(t)-t, Z(t)-t to obtain the following graphs with the following parameters ($\sigma$ =10, r=10, b=8/3.0):

The large value of T is set to have a time step count of STEP = 100000 for large discretized time for observable motion of the Lorenz Attractor. To obtain the below we need all data points isolated by inserting these in the function exec(self):

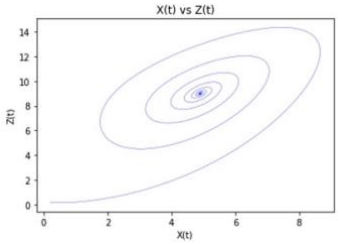
```
dimentioner_res_0 = [i for i in range(len(self.res[0]))]

dimentioner_res_1 = [i for i in range(len(self.res[1]))]

dimentioner_res_2 = [i for i in range(len(self.res[2]))]
```

```
plt.plot(dimentioner_res_2, self.res[2], color = "blue", lw = 0.3)
```

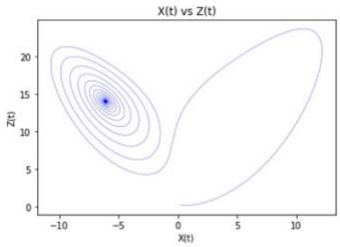
In here, self.res[i = 0, 1, 2] where i = 0, 1, 2 corresponds to the X(t), Y(t), Z(t) data value respectively solved using the RK4 method.



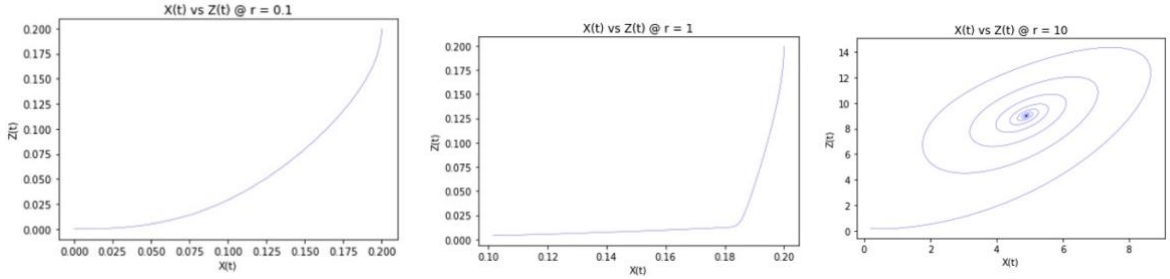

At first glance, we see the trajectories stabilises before 20000th time step. This tells us that the trajectory stabilises at a point in space. This point is called a limit cycle in the Lorenz attractor and if we plot X(t) against Z(t). This is a stable manifold as we observe the trajectory converges into a stable point in space.

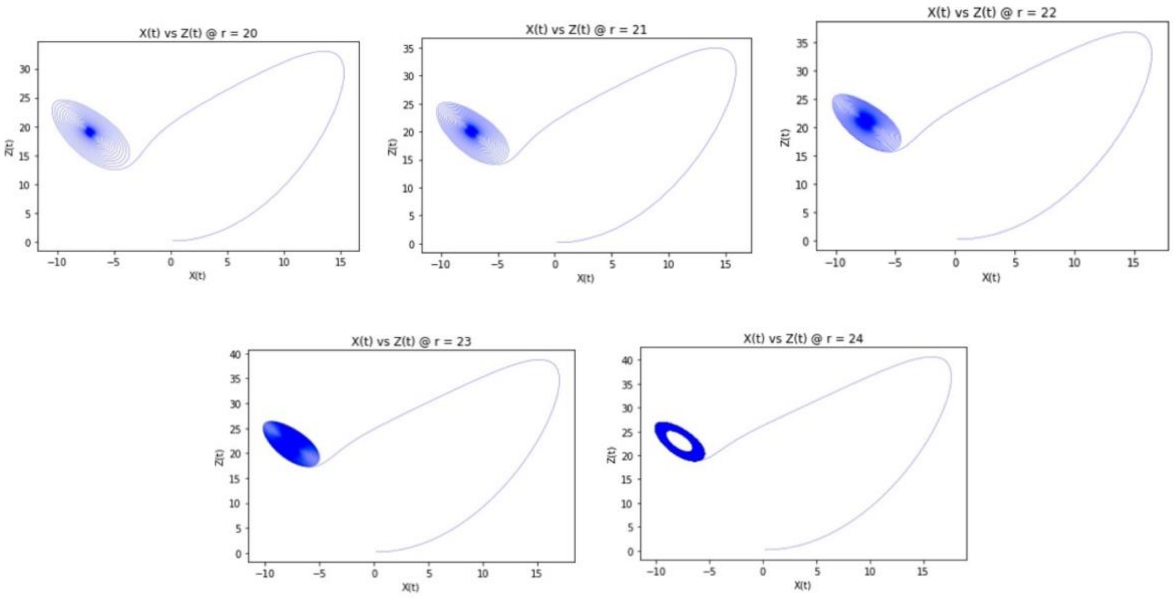



Changing the parameter to r = 15 and the rest same as the previous sketch we get:



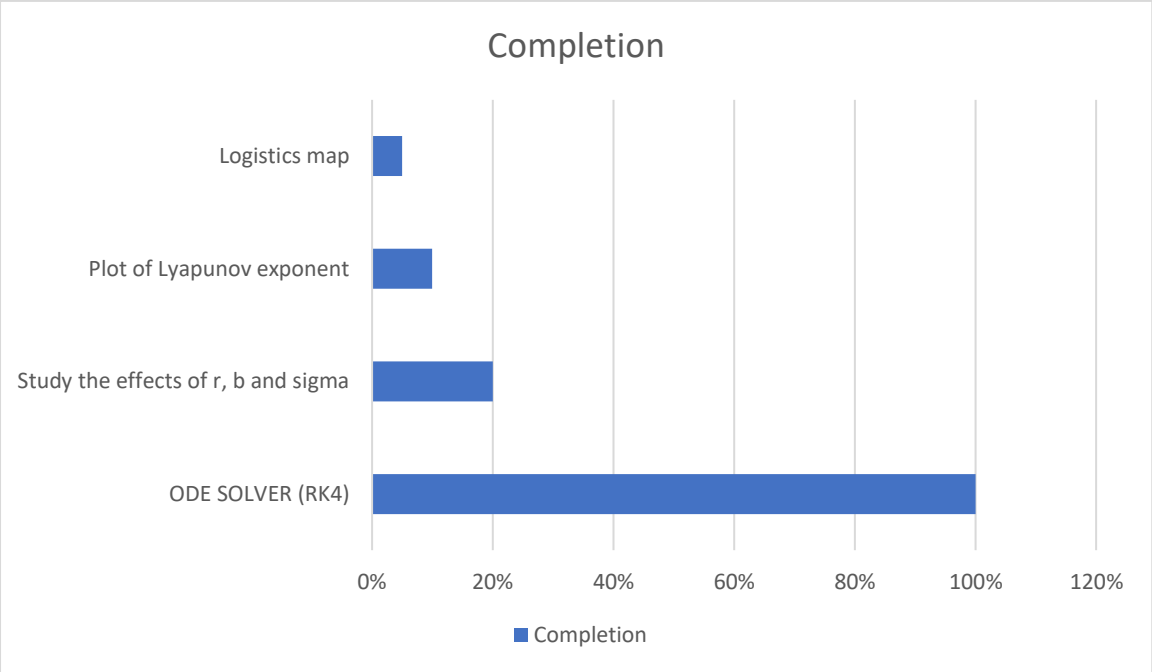

In the following graph, I started to see a small pattern on the formation of trajectory, the circular path of the particle forms beyond r > 10 and decided to crank it up to study it further while keeping the rest constant.





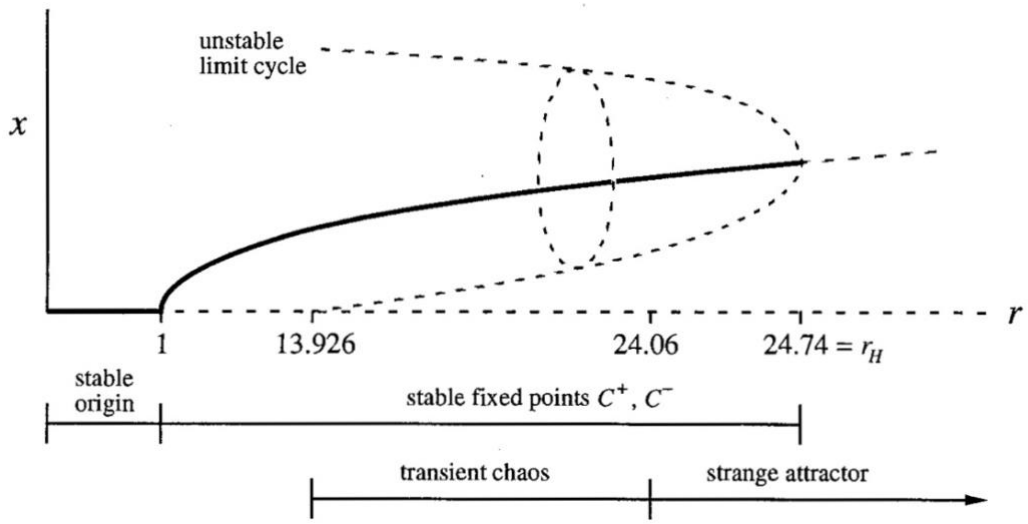

The limit cycle collapses into a dense like path until r >> 24. Below we see the preliminary findings as we approach r >> 24.

# Gantt Chart



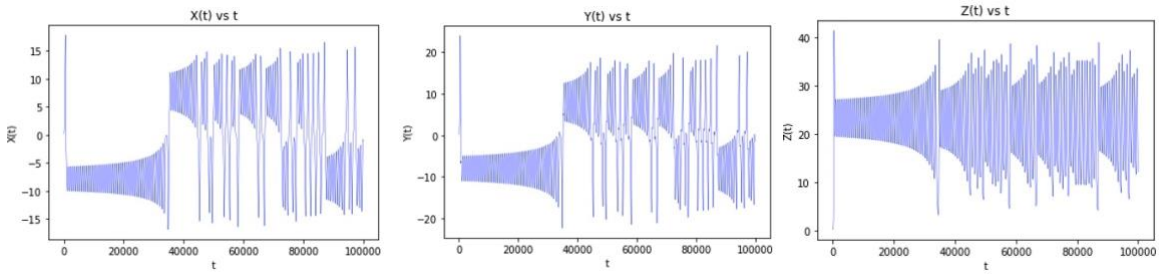

I am still working on the logistics map and the Lyapunov exponent of the logistic map. More study need to be done on the effects of r, b and sigma to show different phases of chaos. Below is a summary of the behaviour of r for the Lorenz attractor trajectory. In the final report we will be studying beyond strange attractor and how it influences the motion of the Lorenz equations in phase space.

The behavior for small values of $r$ is summarized in Figure 9.5.1.



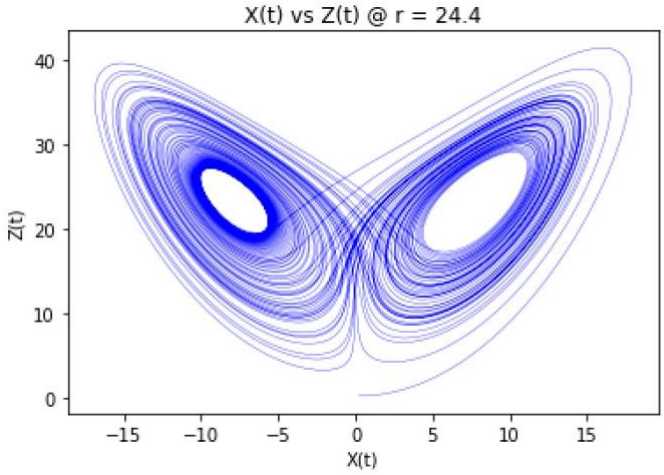

# Preliminary Results

Now, let us plot X(t)-t, Y(t)-t, Z(t)-t to obtain the following graphs with the following parameters ($\sigma$ =10, r=24.4, b=8/3.0):




From the above graphs, we notice a very similar formation of irregular oscillation in the X(t) and Y(t) graph as we increase in time. However, a more visible observation of in the Z(t) graph as we see the amplitude drops significantly after it reaches the peak, in essence, is less irregular in oscillation.

Plotting Z(t) against X(t) we obtain:




We have yet to play around with other parameters and taking extreme conditions, hence in the final report we will be playing with huge numerical values for r >> 100 and r >>1000 to see how the chaos unfolds. Finally, we will try to create a logistic map for the Lorenz equations, plot its Lyapunov exponent and relate back to the Strange Attractor.

# Appendix:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed Mar 11 21:18:55 2020

@author: jameselijah

"""
import sys
import traceback
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from random import randint
import math

class LorenzAttractorRK4:
    DT           = 1e-3     # Differential interval
    STEP         = 100000   # Time step count
    X_0, Y_0, Z_0 = 0.2, 0.2, 0.2  #set innit condition
    #print (X_0, Y_0, Z_0) #Show randomised initial conditions

    def __init__(self):
        self.res = [[], [], []] #initialse the list of data in their respective dimensions

    def exec(self):
        """ Lorenz attractor (RK4 method) execution """
        try:
            xyz = [self.X_0, self.Y_0, self.Z_0]
            for _ in range(self.STEP): #method of RK4
                k_0 = self.__lorenz(xyz)
                k_1 = self.__lorenz([x + k * self.DT / 2 for x, k in zip(xyz, k_0)])
                k_2 = self.__lorenz([x + k * self.DT / 2 for x, k in zip(xyz, k_1)])
                k_3 = self.__lorenz([x + k * self.DT for x, k in zip(xyz, k_2)])
                for i in range(3):
                    xyz[i] += (k_0[i] + 2 * k_1[i] + 2 * k_2[i] + k_3[i]) \
                            * self.DT / 6.0
                    self.res[i].append(xyz[i])

            plt.xlabel("t")
            plt.ylabel("Z(t)")
            plt.title('Z(t) vs t')
            #plt.axes()
            #plt.xlim([0, 50])
            #plt.ylim([-20, 20])
            dimentioner_res_0 = [i for i in range(len(self.res[1]))]
```

```
            plt.plot(self.res[0], self.res[2], color = "blue", lw = 0.3) #plot X, Y
            #plt.plot(self.res[2], self.res[0]) #self.res[0] is values corresponding to x, [1] to y, [2]
to z
            self.__plot() #calls out function to plot in 3D
        except Exception:
            raise

    def __lorenz(self, xyz, p=10, r=24.4, b=8/3.0): #LORENZ PARAMETERS THAT WE
SET
        try:
            return [-p * xyz[0] + p * xyz[1], -xyz[0] * xyz[2] + r * xyz[0] - xyz[1],
                    xyz[0] * xyz[1] - b * xyz[2]]
        except Exception:
            raise

    def __plot(self):
        try:
            fig = plt.figure()
            ax = Axes3D(fig) #use of 3D to see nice butterfly effect
            ax.set_xlabel("x")
            ax.set_ylabel("y")
            ax.set_zlabel("z")
            #ax.title ("3D plot of the particle")
            ax.plot(self.res[0], self.res[1], self.res[2], color="red", lw=0.3)

        except Exception:
            raise

    def plot(self):
        plt.plot(self.res[0], self.res[1], colour = "blue", lw=1)
        plt.show()

if __name__ == '__main__':
    try:
        obj = LorenzAttractorRK4()
        obj.exec()

    except Exception as e:
        traceback.print_exc()
        sys.exit(1)
```

# Reference

Non-linear Dynamics and Chaos with applications to Physics, Biology, Chemistry and Engineering by Steven H. Strogatz