104283 - Introduction to Numerical Analysis
Spring 2023
Python Assignment 4

Name: Haoyi Yang
Student ID: 999009798

June 14, 2023

Runge-Kutta Integrator

1. Question

Write the following function in Python: $RK4\_step(func, dt, tk, wk)$
The function gets the following parameters as input:
- Some function - $func$ (Python function).
- Time step - dt.
- Time value – tk.
- Function state – wk.

The function calculates a single RK4 step and returns: (t_{k+1}, w_{k+1}). Meaning, the next time step and its corresponding function state evaluation at that time.
You may use Algorithm 5.2 (page 288) for reference.

2. Code and Explanation

We can mainly follow the alogithm 5.2 in the page 288 to write this algorithm. Just modify some parameters:

```
def RK4_step(func, dt, tk, wk):
    K1 = dt * func(tk, wk)
    K2 = dt * func(tk + dt / 2, wk + K1 / 2)
    K3 = dt * func(tk + dt / 2, wk + K2 / 2)
    K4 = dt * func(tk + dt, wk + K3)

    w = wk + (K1 + 2 * K2 + 2 * K3 + K4)/6 # This w is w_k+1.
    t = tk + dt  # This t is t_k+1.
    return(t,w)
```

Lorenz Equations

1. Question

The Lorenz equations are a system of three coupled non-linear ordinary differential equations developed by Edward Lorenz in 1963 to study the dynamics of weather patterns:

$$\begin{cases} \dfrac{dx}{dt} = \sigma(y - x) \\ \dfrac{dy}{dt} = x(\rho - z) - y \\ \dfrac{dz}{dt} = xy - \beta z \end{cases} \tag{1}$$

The constants σ, ρ, and β are parameters that determine the behavior of the system. To observe chaotic behavior, a common choice is: $\sigma = 10$, $\beta = \frac{8}{3}$, $\rho = 28$.
Write the following function in Python:
$lorenz(t, w)$
The function gets the time t and state $w = x, y, z$ as input arguments. The function returns the set of values $(\frac{dx}{dt}, \frac{dy}{dt}, \frac{dz}{dt})$. Use the suggested values for σ, ρ, and β.

2. Code and Explanation

It's really easy to generate this function, just plug the values to the equation (1) so we get the function. Remember that we need to use numpy array, so we import numpy first:

```
1 import numpy as np
2 def lorenz(t,w):
3     x,y,z = w
4     return np.array([10 * (y - x), x * (28 - z) - y, x * y - (8/3) * z])
```

3D Plot

1. Question

In a chaotic system, a small change in initial conditions will result in different behaviors which are magnified over time. Solve the equations for two different initial conditions using the functions you wrote. Create a 3D plot of the two results on the same figure, allowing you to observe the divergence of trajectories over time. Choose time step $dt = 0.01$ and any pair of initial states (x_0, y_0, z_0). Modify the total solution time so that the results are clear.

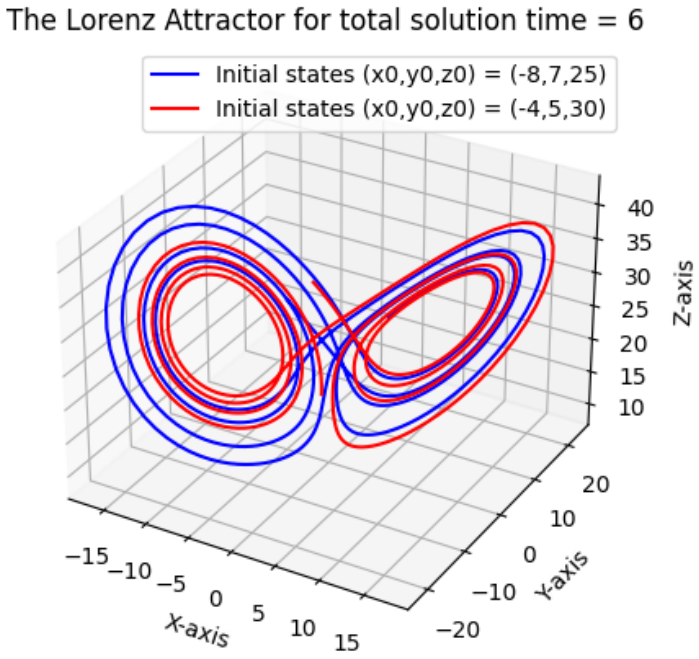
2. Code, Explanation and Results

We first choose two different initial states x_0, y_0, z_0 (I selected many data pairs and found that these two looks better): $(-8, 7, 25)$ and $(-4, 5, 30)$. Then we first set the total solution time equal to 6. And here's my code (the functions inside are defined above):

```
import matplotlib.pyplot as plt
w0 = np.array([-8, 7, 25])
w00 = np.array([-4, 5, 30])
dt = 0.01
t = 0
total_solution_time = np.arange(0.0, 6.0, dt)
# We choose time = 6 here.
state1 = []
state2 = []
wk1 = w0
wk2 = w00

for i in total_solution_time:
    state1.append(wk1)
    wk1 = RK4_step(lorenz, 0.01, i, wk1)[1]
state1 = np.array(state1)
for j in total_solution_time:
    state2.append(wk2)
    wk2 = RK4_step(lorenz, 0.01, j, wk2)[1]
state2 = np.array(state2)

# Create figure and 3D axes:
fig = plt.figure()
ax = plt.axes(projection='3d')
x_data1 = []
y_data1 = []
z_data1 = []
x_data2 = []
y_data2 = []
z_data2 = []
for m in range(len(state1)):
    x_data1.append(state1[m][0])
    y_data1.append(state1[m][1])
    z_data1.append(state1[m][2])
for n in range(len(state2)):
    x_data2.append(state2[n][0])
    y_data2.append(state2[n][1])
    z_data2.append(state2[n][2])

```

```
# Add the plots:
ax.plot3D(x_data1, y_data1, z_data1, 'b', label="Initial states (x0,y0,z0) = (-8,7,25)")
ax.plot3D(x_data2, y_data2, z_data2, 'r', label="Initial states (x0,y0,z0) = (-4,5,30)")
plt.title("The Lorenz Attractor for total solution time = 6")
ax.set_xlabel('X-axis')
ax.set_ylabel('Y-axis')
ax.set_zlabel('Z-axis')
plt.legend()
plt.show()
```

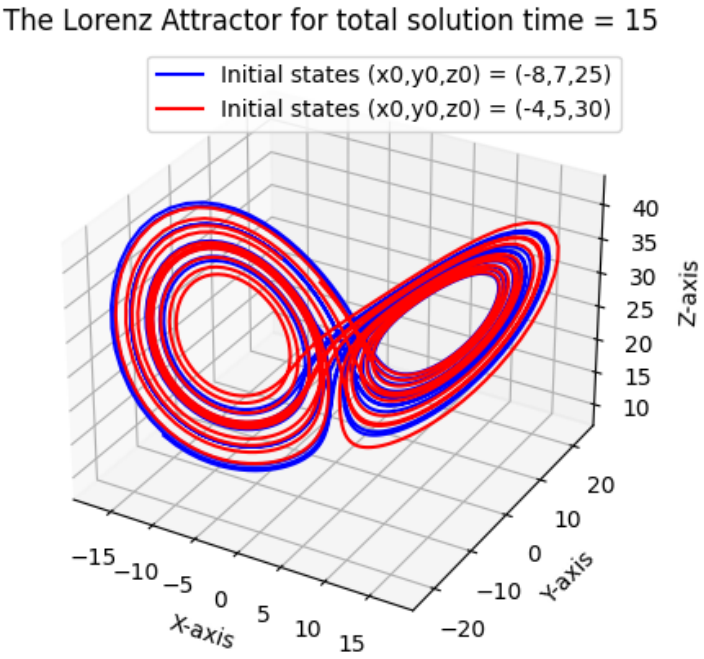
And we get the result 3D figure for total solution time = 6:



Then we modify this code to let the total solution time be 15:

```
total_solution_time = np.arange(0.0, 15.0, dt)
```

Then we can get another result 3D figure for total solution time = 15, we can see the results clearly by comparing the two figures:


The Lorenz Attractor for total solution time = 15
Initial states (x0,y0,z0) = (-8,7,25)
Initial states (x0,y0,z0) = (-4,5,30)
X-axis
Y-axis
Z-axis
−15
−10
−5
0
5
10
15
−20
−10
0
10
20
10
15
20
25
30
35
40