Assignment

The Lorentz equations

Name TN 1: Tim Peinkofer

tim.peinkofer@student.uni-tuebingen.de

Name TN 2: Fabian Kostow

fabian.kostow@student.uni-tuebingen.de

Tutor: Jose Carlos Olvera Meneses

Date: 31. Januar 2025

Inhaltsverzeichnis

1 Lorentz equations

1.1 Animation of the Lorentz equations

In the following we want to solve the Lorentz equations:

$$
\begin{aligned}
x' &= \sigma \cdot (y - x) \\
y' &= x \cdot (R - z) - y \\
z' &= xy - \beta z \\
\beta &= \frac{8}{3} \\
R &= 28 \\
\sigma &= 10
\end{aligned}
$$

If we apply Runge-Kutta-4th order on the ODE system with initial condition (x,y,z)(0)=(1,1,1) we get:

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib.animation import FuncAnimation

def rk4(system, y0, t):
    n = len(t)
    h = t[1] - t[0]  # Calculate step size
    y = np.zeros((n, len(y0)))
    y[0] = y0

    for i in range(1, n):
        k1 = system(y[i - 1], t[i - 1])
        k2 = system(y[i - 1] + h * k1 / 2, t[i - 1] + h / 2)
        k3 = system(y[i - 1] + h * k2 / 2, t[i - 1] + h / 2)
        k4 = system(y[i - 1] + h * k3, t[i - 1] + h)
        y[i] = y[i - 1] + h * (k1 + 2 * k2 + 2 * k3 + k4) / 6
    return y

def lorenz_attractor(r, t):
    sigma, R, beta = 10, 28, 8 / 3
    x, y, z = r
    return np.array([
        sigma * (y - x),
        x * (R - z) - y,
        x * y - beta * z
    ])
```

```
# Time parameters
t0, t_end, h = 0, 50, 1e-3
t = np.linspace(t0, t_end, int((t_end - t0) / h) + 1)
y0 = np.array([1, 1, 1])

# Compute the solution
r = rk4(lorenz_attractor, y0, t)

# Create 3D animation
fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')
ax.set_title("Lorenz Attractor")
ax.set_xlabel("X")
ax.set_ylabel("Y")
ax.set_zlabel("Z")

# Set axis limits
ax.set_xlim(-25, 25)
ax.set_ylim(-35, 35)
ax.set_zlim(0, 50)

# Initialize line and point
line, = ax.plot([], [], [], lw=0.5, color='b')
point, = ax.plot([], [], [], 'ro')

# Add a text object for time display
time_template = 'Time = %.1f'
time_text = ax.text2D(0.05, 0.95, '', transform=ax.transAxes)

def init():
    line.set_data([], [])
    line.set_3d_properties([])
    point.set_data([], [])
    point.set_3d_properties([])
    time_text.set_text('')
    return line, point, time_text
```

```
def update(frame):
    if frame >= len(r):
        raise ValueError("frame index is out of bounds")
    line.set_data(r[:frame, 0], r[:frame, 1])
    line.set_3d_properties(r[:frame, 2])
    point.set_data([r[frame, 0]], [r[frame, 1]])  # Passing
                                    sequences
    point.set_3d_properties([r[frame, 2]])        # Passing a
                                     sequence
    time_text.set_text(time_template % (frame * h))
    ax.view_init(elev=30, azim=frame * 0.1)
    return line, point, time_text

# Set the interval to 1 ms to achieve 1 second per real
                                  second
ani = FuncAnimation(fig, update, frames=len(r), init_func=
                                  init, interval=1, repeat=False
                                  )
plt.show()
```

The result is the following:

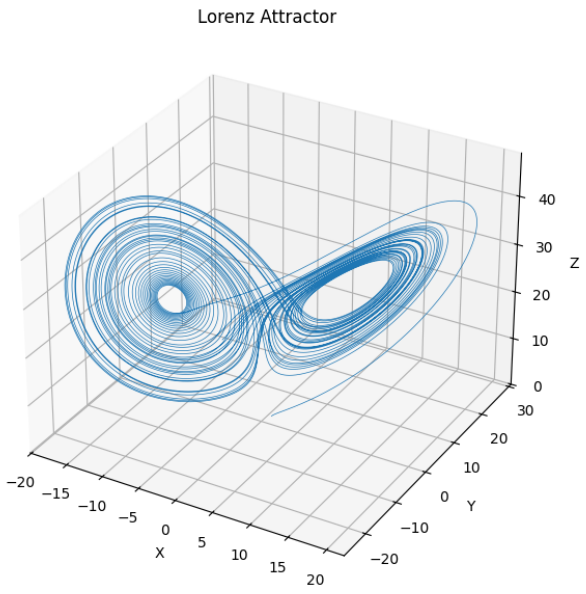

Abbildung 1: Lorentz equations

1.2 Chaotic bevaior

We now want to take a look at the chaotic behavior of the Lorentz equations for slightly different initial values. To compare the results for the initial guesses (x,y,z)(0)=(1,1,1) and (x,y,z)(0)=(1+1e-9,1,1) we plot the x-values for both guesses:

```
import numpy as np
import matplotlib.pyplot as plt

def rk4(System, y0, t, h):
    n = len(t)
    y = np.zeros((n, len(y0)))
    y[0] = y0

    for i in range(1, n):
        k1 = System(y[i - 1]) * h
        k2 = System(y[i - 1] + k1 / 2) * h
        k3 = System(y[i - 1] + k2 / 2) * h
        k4 = System(y[i - 1] + k3) * h
        y[i] = y[i - 1] + (k1 + 2 * k2 + 2 * k3 + k4) / 6

    return y

def Lorenz_Attractor(r):
    sigma = 10
    R = 28
    beta = 8 / 3
    x, y, z = r
    Xprime = sigma * (y - x)
    Yprime = x * (R - z) - y
    Zprime = x * y - beta * z

    return np.array([Xprime, Yprime, Zprime])

t0, t_end = 0, 50
h = 1e-3
t = np.arange(t0, t_end, h)
y0 = np.array([1, 1, 1])
y1 = np.array([1 + 1e-9, 1, 1])

r = rk4(Lorenz_Attractor, y0, t, h)
r_1 = rk4(Lorenz_Attractor, y1, t, h)
```

```
# Plot the 3D trajectory
plt.figure(figsize=(10, 7))
plt.plot(t, r[:, 0], color="g", label="Initial Condition: (1,
                                    1,1)")
plt.plot(t, r_1[:, 0], color="b", label="Perturbed: (1+1e-9,1
                                    ,1)")
plt.title("Lorenz Attractor - X Component Over Time")
plt.xlabel("Time t")
plt.ylabel("x(t)")
plt.legend()
plt.show()
```

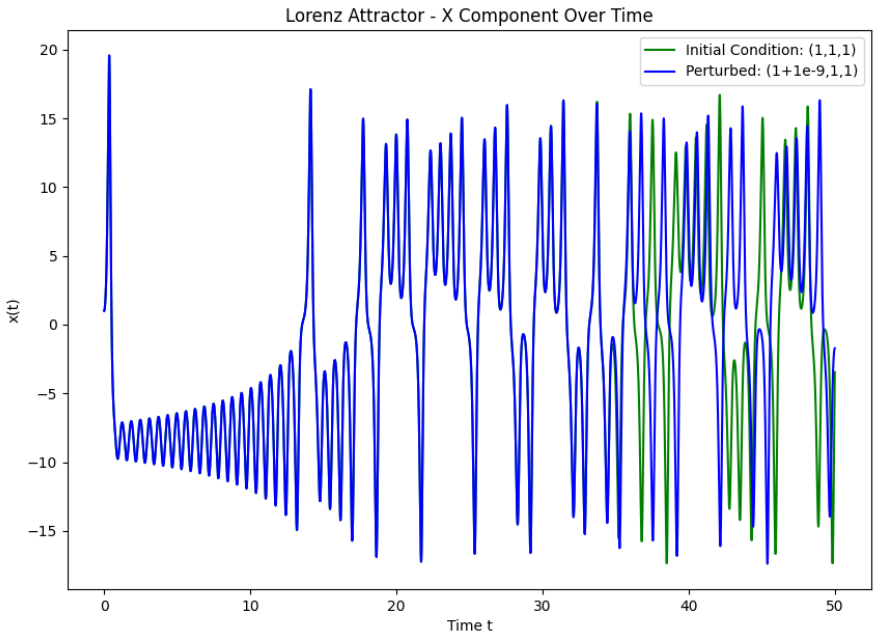

Abbildung 2: Chaotic behavior

As we see the trajectories separate at t = 37 s . As we see the ODE system is very sensitive to initial guesses.