بررسی کد چرخه آب آشفته

سعید براری

۹۸۱۳۱۸

saeedbarari@iasbs.ac.ir

فراخوانی کتابخانه‌ها:

```python
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
from scipy.integrate import solve_ivp
from datetime import datetime
```

۱) کتابخانه numpy برای محاسبات ریاضی استفاده شده است

۲) matplotlib.pyplot برای رسم نمودار فراخوانی شد

۳) matplotlib.animation برای ساخت انیمیشن از حرکت دو ذره با شرایط اولیه نزدیک بهم و دیدن آشوب استفاده شده است

۴) scipy.integrate برای حل معادلات دیفرانسیل به روش رونگه کوتا ۴۵ فراخوانی گردید

پارامترهای معادلات لورنز:

```python
# Parameters for the Lorenz system
sigma = 10.0
beta = 8.0 / 3.0 # Geometric factor
rho = 28.0          # Rayleigh number
```

در این بخش سه پارامتر معادلات لورنز را مشخص نمودم و مقدار عدد ریلی را برابر با ۲۸ گرفم تا در منطقه حرکت آشوبناک قرار بگیرد که زیرا از $r_H = ۲۴.۷۵$ بیشتر است ($\sigma = ۱۰$ و $b = \frac{۸}{۳}$)

$$r_H = \frac{\sigma(\sigma+b+۳)}{\sigma-b-۱} \cong ۲۴.۷۵$$

مکان اولیه دو ذره:

```python
# Initial conditions for two trajectories
init_1 = [1.0, 1.0, 1.0]
init_2 = [1.1, 1.0, 1.0]   # Slight perturbation
```

برای این که حرکت آشوبناک معادلات لورنز را مشاهده کنیم دو ذره با شرایط اولیه نزدیک به هم انتخاب میکنیم و سپس مشاهده میکنیم که ذرات در گذر زمان در ابتدا طبق پیش بینی حرکت میکنند اما پس از گذشت مدت زمانی از یکدیگر فاصله میگیرند و مسیرهای متفاوتی را در پیش میگیرند. در این قسمت مختصات اولیه ذرات را مشخص کردم به ترتیب مولفه‌های ۱، ۲ و ۳ مولفه در راستاهای x, y, z می‌باشد.

تعریف مدت زمان انجام آزمایش:

```python
start = 0
finish = 40
time_step = 5000
```

در این مسئله به ذرات اجازه دادیم که از زمان صفر تا ۴۰ واحد زمان در گام های $\frac{۱}{۱۲۵}$ واحد زمانی حرکت کنند.

تعریف معادلات لورنز:

در کلاس دیدیم که دینامیک چرخه آب آشفته را میتوانیم با استفاده از معادلات لونز نشان دهیم در این بخش از کد معادلات لورنز که به فرم زیر هستند را به کامپیوتر دادم.

```python
def lorenz_equations(t, location, sigma, beta, rho):
    x, y, z = location

    dx_dt = sigma * (y - x)
    dy_dt = x * (rho - z) - y
    dz_dt = x * y - beta * z

    return [dx_dt, dy_dt, dz_dt]
```

یادآوری: معادلات لورنز سه معادله هستند که به فرم زیر می‌باشد

$$\frac{dx}{dt} = \sigma(y - x) \qquad \frac{dy}{dt} = x(\rho - z) - y \qquad \frac{dz}{dt} = xy - bz$$

تعریف گام‌های زمانی:

در این بخش با توجه به مولفه‌های زمانی گام‌های زمانی را تعریف کردم

```python
t_span = (start, finish)
t_eval = np.linspace(t_span[0], t_span[1], time_step)
```

حل معادلات دیفرانسیل:

با استفاده از روش رونگه کوتا ۴۵، معادلات لورنز را با توجه به گام های زمانی که در بخش قبل تعریف کردم حل کردم و برای دو ذره ۱ و ۲ ذخیره نمودم

```python
sol1 = solve_ivp(lorenz_equations,
                 t_span, init_1,
                 args=(sigma, beta, rho),
                 t_eval=t_eval, method='RK45')

sol2 = solve_ivp(lorenz_equations,
                 t_span, init_2,
                 args=(sigma, beta, rho),
                 t_eval=t_eval, method='RK45')
```

یادآوری:

رونگه کوتا به دسته‌ای از مهم‌ترین روشهای حل عددی معادلات دیفرانسیل عادی گفته می‌شود که توسط دو دانشمند آلمانی، رونگه و کوتا ابداع شده است. روشی که در این مسئله استفاده شد *RK45* است. از یک تابع از پیش نوشته شده در پایتون از کتابخانه *scipy.integrate* استفاده نمودم.

رسم نمودار انیمیشن:

برای رسم نمودار در ابتدا ویژگی ظاهری کلی نمودار را مشخص نمودم

```python
# Set up the figure and 3D axis
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')
ax.set_xlim((min(np.min(x1), np.min(x2)), max(np.max(x1), np.max(x2))))
ax.set_ylim((min(np.min(y1), np.min(y2)), max(np.max(y1), np.max(y2))))
ax.set_zlim((min(np.min(z1), np.min(z2)), max(np.max(z1), np.max(z2))))
ax.set_title("Lorenz System (Chaotic Waterwheel Evolution)")
ax.set_xlabel("x")
ax.set_ylabel("y")
ax.set_zlabel("z")
```

سپس ویژگی نمودار هر ذره و خط مسیر حرکت آن را مشخص کردم

```python
# Initialize lines and points for animation
line1, = ax.plot([], [], [], lw = 0.8, color="blue",
                    label=f"({init_1[0]}, {init_1[1]}, {init_1[2]})")
point1, = ax.plot([], [], [], 'o', color="red")
line2, = ax.plot([], [], [], lw=0.8, color="green",
                    label=f"({init_2[0]}, {init_2[1]}, {init_2[2]})")
point2, = ax.plot([], [], [], 'o-', color="orange")
```

بعد از آن برای نمایش انیمیشنی حرکت دو ذره از دو تابع ***init*** و ***update*** استفاده کردم (این بخش از کد را چون تا حالا انیمیشنی کد نزدم از ***chat gbt*** گرفتم اما به طور کلی تابع ***init*** مشخص کننده یک ساختار کلی است و تابع ***update*** در هر فریم که همان تعداد گام زمانی است مختصات ذره را ثبت می‌کند.

```python
# Initialization function for animation
def init():
    line1.set_data([], [])
    line1.set_3d_properties([])
    point1.set_data([], [])
    point1.set_3d_properties([])

    line2.set_data([], [])
    line2.set_3d_properties([])
    point2.set_data([], [])
    point2.set_3d_properties([])

    return line1, point1, line2, point2
```

```python
# Update function for animation
def update(frame):
    line1.set_data(x1[:frame], y1[:frame])
    line1.set_3d_properties(z1[:frame])
    point1.set_data(x1[frame], y1[frame])
    point1.set_3d_properties(z1[frame])

    line2.set_data(x2[:frame], y2[:frame])
    line2.set_3d_properties(z2[:frame])
    point2.set_data(x2[frame], y2[frame])
    point2.set_3d_properties(z2[frame])

    return line1, point1, line2, point2
```

در آخر تعداد گام های زمانی را به عنوان فریم می‌دهیم و با استفاده از تابع *FuncAnimation* از کتابخانه *matplotlib.animation* انیمیشن را ساختم.

```
frames = len(t_eval)
ani = FuncAnimation(fig, update,
                    frames=frames,
                    init_func=init,
                    nterval=20, blit=True)
```

رسم نمودار:

در این بخش ویژگی نمودار تحول زمانی مکان ذرات را نوشتم

```
fig = plt.figure(figsize=(14, 8))
# 3D plot of the Lorenz attractor
ax = fig.add_subplot(121, projection='3d')
ax.plot(x1, y1, z1, lw=0.7,
        label=f"({init_1[0]}, {init_1[1]}, {init_1[2]})",
        color = "blue")
ax.plot(x2, y2, z2, lw=0.7,
        label=f"({init_2[0]}, {init_2[1]}, {init_2[2]})",
        linestyle='dashed',color = "green")
ax.set_title("Lorenz System (Chaotic Waterwheel)")
ax.set_xlabel("x")
ax.set_ylabel("y")
ax.set_zlabel("z")
ax.legend()
```

این بخش از کد نیز برای نشان دادن آشفتگی سیستم و برای یافتن نمای لیاپانوف میباشد و آن را ذخیره کردم.

```
# Time series to highlight butterfly effect
ax2 = fig.add_subplot(122)
delta_r = np.sqrt((x1-x2)**2+(y1-y2)**2+(z1-z2)**2)
ax2.plot(t_eval, delta_r, lw=1.2, color='red')
ax2.set_title("Divergence of Nearby Trajectories")
ax2.set_xlabel("Time")
ax2.set_ylabel("|r1 - r2|")
ax2.set_yscale('log')
plt.tight_layout()
plt.savefig(f"/home/sb/Desktop/{datetime.now()}.png")
plt.show()
```

نتایج:

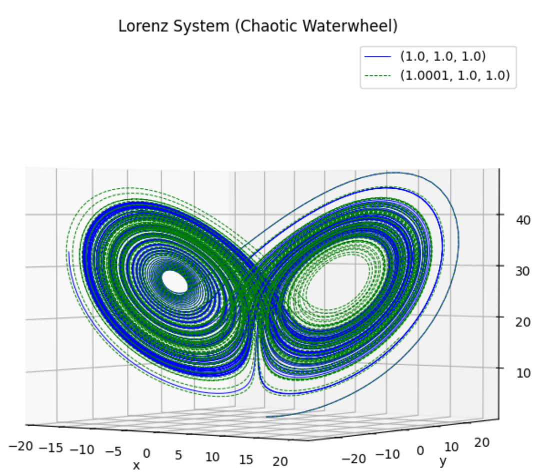
در این گزارش دو حالت $\rho = ۲۸$ و $\rho = ۱۶۰$ را بررسی کردم اما میتوان با تغیر مقادیر خطوط ۷ تا ۱۷ شرایط دیگر را نیز بررسی کرد

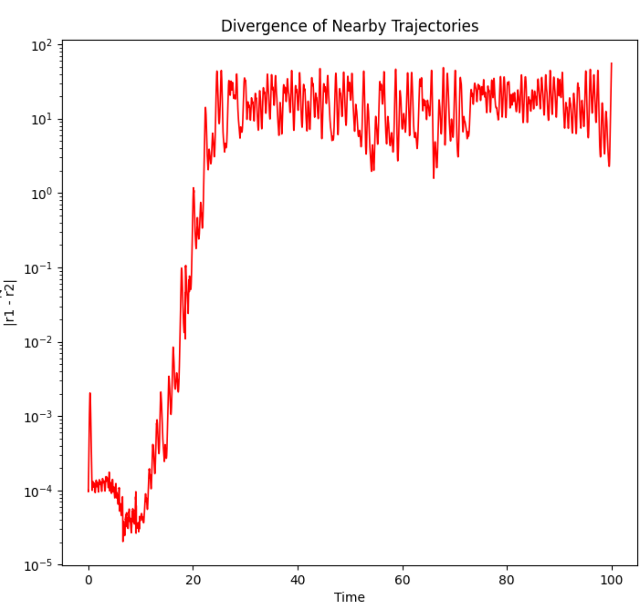
شرایط اولیه ای که در این گزارش بررسی شد برابر است با:

- مختصات ذره اول (۱، ۱، ۱) و مختصات ذره دوم (۱، ۱، ۱.۰۰۰۱) – همانطور که مشاهده می‌کنید تنها اختلاف مکانی این دو ذره در راستای x و از مرتبه $۱۰^{-۴}$ می‌باشد
- $b = \frac{۸}{۳}$ و $\sigma = ۱۰$
- $t_i = ۰$ ، $t_f = ۱۰۰$ و $dt_i = \frac{۱}{۱۵۰}$ میباشد

نتیجه برای $\rho = ۲۸$:

همانطور که در تصویر قابل ملاحظه است حرکت دو ذره شبیه بال‌های پروانه است که به این شکل الگوی پروانه‌ای می‌گویند و این مورد را لورنز به عنوان اثر پروانه‌ای مطرح کرده است. نمودار سمت راست نمودار لگاریتمی $|\delta(t)|$ برحسب t می‌باشد. شیب مثبت نمودار سیستم آشفته میباشد. ویدیوی تحول حرکت دو ذره در تلگرام ارسال شد.





نتیجه برای $\rho =$ ۱۶۰:

همان طور که در ارائه صحبت کردم و دیدیم برای حالت $\rho =$ ۱۶۰ یک چرخه پایدار است که دیگر ویدیو آن را ضبط نکردم اما با تغیر ρ این ویدیو را مشاهده کنیم . در تصویر پایین چرخه پررنگ سبز زیر همان چرخه پایدار است.