Imports

```
import numpy as np
import matplotlib.pyplot as plt
from tqdm.auto import tqdm

sigma = 10
r = 28
b = 8/3

def lorenz_step(t, x1, x2, x3, dt, sigma, rho, beta):
    """Calculate the next step in the Lorenz system."""
    dx1 = sigma * (x2 - x1) * dt
    dx2 = (x1 * (rho - x3) - x2) * dt
    dx3 = (x1 * x2 - beta * x3) * dt

    return t + dt, x1 + dx1, x2 + dx2, x3 + dx3

def lorenz(t0, x1_0, x2_0, x3_0, dt, sigma, rho, beta, iter_num):
    """Calculate the evolution of the Lorenz system."""
    t = np.zeros(iter_num + 1)
    x1 = np.zeros(iter_num + 1)
    x2 = np.zeros(iter_num + 1)
    x3 = np.zeros(iter_num + 1)

    t[0], x1[0], x2[0], x3[0] = t0, x1_0, x2_0, x3_0
    for i in range(iter_num):
        t[i + 1], x1[i + 1], x2[i + 1], x3[i + 1] = (
            lorenz_step(t[i], x1[i], x2[i], x3[i], dt, sigma, rho,
beta)
        )

    return t, x1, x2, x3
```

Generate training set

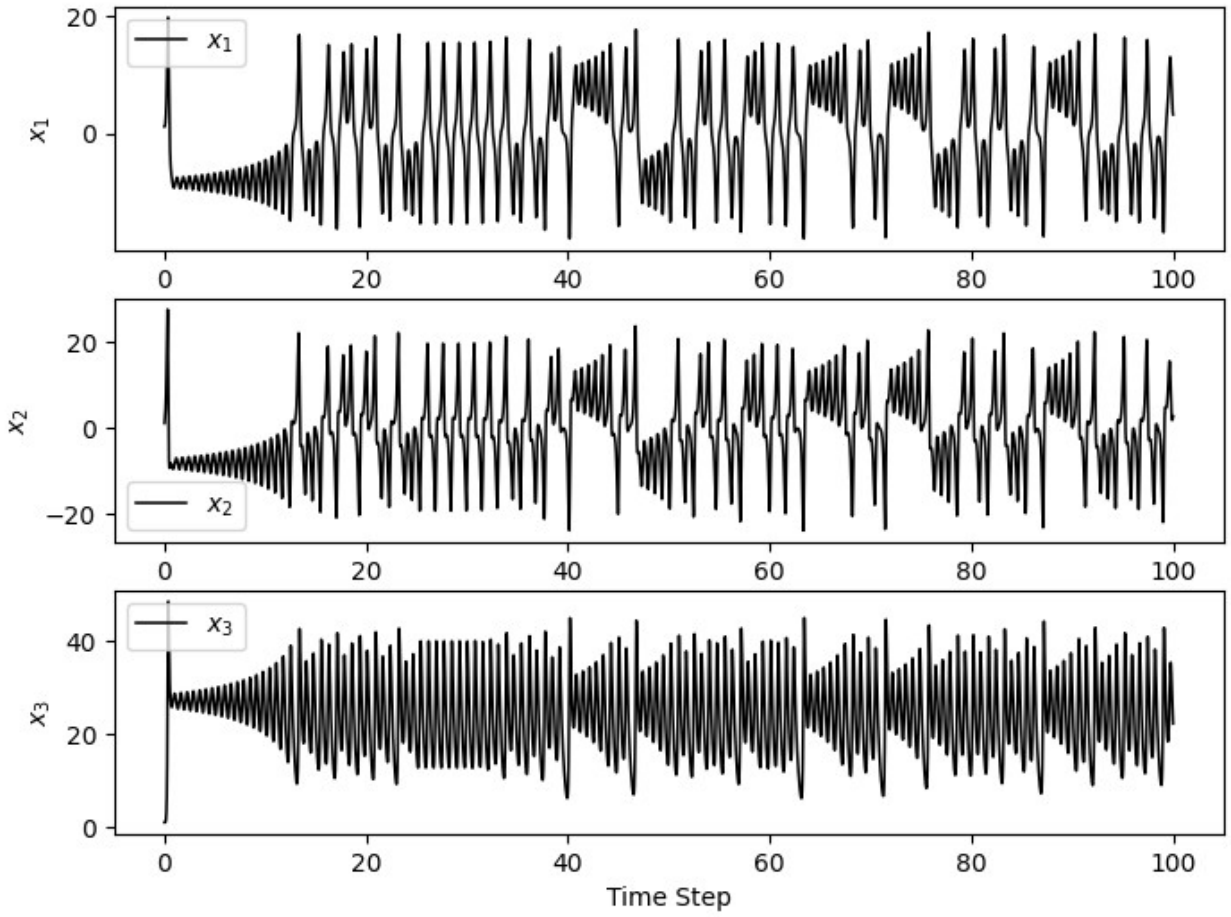
```
num_series = 1
iter_num = 100000
dt = 0.001
std = 0.05
series = []
for _ in tqdm(range(num_series)):
    t, x1, x2, x3 = lorenz(t0=0, x1_0=np.random.normal(1, std),
x2_0=np.random.normal(1, std), x3_0=np.random.normal(1, std), dt=dt,
                           sigma=10, rho=28, beta=8 / 3,
iter_num=iter_num)
    series.append([x1, x2, x3])
series = np.array(series)
```

{"model_id":"eca5b347b1f343858e2f6cac3faa926e","version_major":2,"version_minor":0}

```
x1 = series[0, 0]
x2 = series[0, 1]
x3 = series[0, 2]
x_zip = list(zip(x1, x2, x3))
X = np.array(x_zip)

fig, axs = plt.subplots(3, 1, figsize=(8, 6))
X = np.array(x_zip)
for i in range(3):
    axs[i].plot(t, X[:,i], label=fr"$x_{i+1}$", color="k", lw=1.1)
    axs[i].set_ylabel(fr"$x_{i+1}$")
    axs[i].legend()
axs[2].set_xlabel("Time Step")
```

```
Text(0.5, 0, 'Time Step')
```

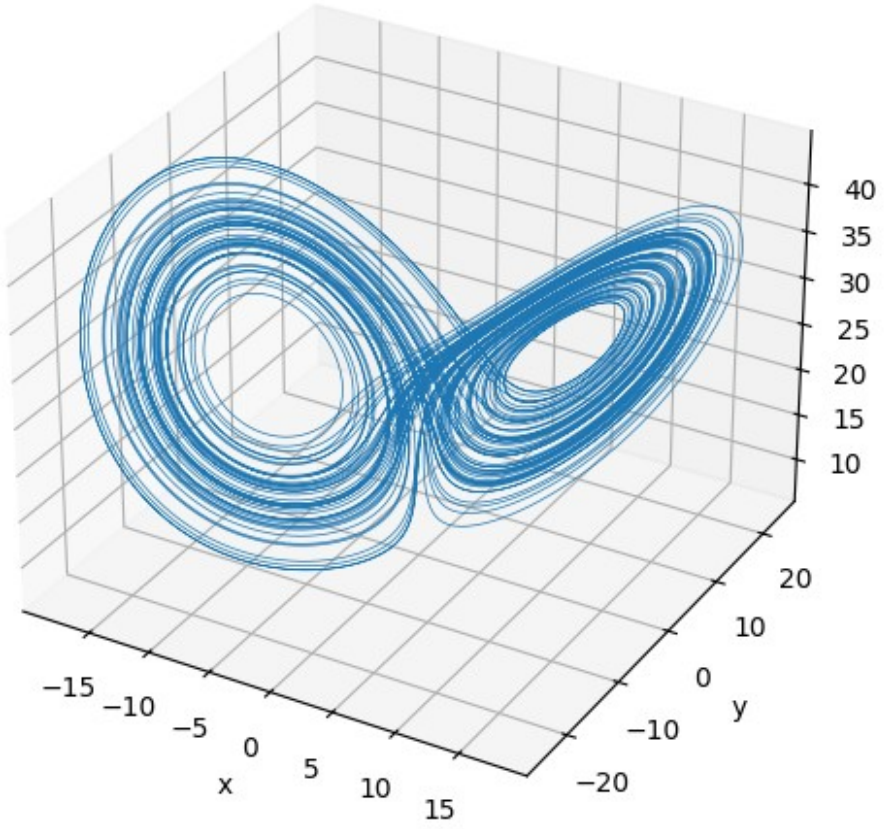



```
fig= plt.figure(figsize=(8, 6))
ax = fig.add_subplot(projection="3d")
```

```
initialpart = int(0.3*iter_num)
ax.plot(x1[initialpart:], x2[initialpart:], x3[initialpart:], lw=0.5)

ax.set_xlabel("x")
ax.set_ylabel("y")
ax.set_zlabel("z")

plt.show()
```




```
def jac(x1, x2, x3, sigma, r, b):
    return np.array([[-sigma, sigma, 0], [r-x3, -1, -x1], [x2, x1, -
b]])

Q = np.eye(3)
Id = np.eye(3)
lambdas = np.zeros(3)

for n in range(len(t)):
    x1s, x2s, x3s = X[n,0], X[n,1], X[n,2]

    M = Id + jac(x1s, x2s, x3s, sigma, r, b)*dt
```

```
    Q, R = np.linalg.qr(np.matmul(M, Q))

    lambdas[0] += np.log(np.abs(R[0, 0]))
    lambdas[1] += np.log(np.abs(R[1, 1]))
    lambdas[2] += np.log(np.abs(R[2, 2]))

lambdas = lambdas/(len(t)*dt)
print(f'Lyapunov exponents in x1: {lambdas[0]:.2f}, in x2:
{lambdas[1]:.2f} and x3: {lambdas[2]:.2f}')
print(f'The largest Lyapunov exponent is found to be approximatly
{max(lambdas):.2f}')
```

```
Lyapunov exponents in x1: 0.83, in x2: -0.01 and x3: -14.52
The largest Lyapunov exponent is found to be approximatly 0.83
```

```
theortic_sum = -(sigma+1+b)
simulation_sum = np.sum(lambdas)
print(f"Theoretic Sum of Lyapunov Exponents: {theortic_sum:.2f}")
print(f"Simulation Sum of Lyapunov Exponents: {simulation_sum:.2f}")
```

```
Theoretic Sum of Lyapunov Exponents: -13.67
Simulation Sum of Lyapunov Exponents: -13.70
```