Lorenz.py

```python
#%% Import libraries
import numpy as np
from sympy import symbols, Eq, solve, Matrix, trace
from scipy.integrate import solve_ivp
import matplotlib.pyplot as plt

#%% Functions
def lorenz(t, state, sigma, r, b):
    x, y, z = state
    dxdt = sigma * (y - x)
    dydt = r * x - y - x * z
    dzdt = x * y - b * z
    return [dxdt, dydt, dzdt]

#%% a)
sigma, b, r = 10, 8/3, 28
x, y, z = symbols('x y z')

f1 = sigma * (y - x)
f2 = r * x - y - x * z
f3 = x * y - b * z

fixed_points = solve([Eq(f1, 0), Eq(f2, 0), Eq(f3, 0)], (x, y, z))

num_fixed_points = len(fixed_points)

num_stable = 0
for point in fixed_points:
    J = Matrix([[sigma * (-1), sigma, 0],
                [r - point[2], -1, -point[0]],
                [point[1], point[0], -b]])

    eigenvalues = J.eigenvals()

    if all(ev.as_real_imag()[0] < 0 for ev in eigenvalues):
        num_stable += 1

result = [num_fixed_points, num_stable]
print(result)

# %% b)

initial_state = [0.1, 0.1, 0.1]

# Time span for the integration
t_span = (0, 50)
t_eval = np.linspace(t_span[0], t_span[1], 10000)

solution = solve_ivp(lorenz, t_span, initial_state, t_eval=t_eval, args=(sigma, r, b))

# Discard the initial part of the solution to focus on the steady state
```

```python
discard = 1000
x, y, z = solution.y[:, discard:]

fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')
ax.plot(x, y, z, lw=0.5)
ax.set_title("Lorenz Attractor")
ax.set_xlabel("X")
ax.set_ylabel("Y")
ax.set_zlabel("Z")
plt.show()

# %% c)
x, y, z, r, b, sigma = symbols('x y z r b sigma')
F1 = sigma * (y - x)
F2 = r * x - y - x * z
F3 = x * y - b * z

F = Matrix([F1, F2, F3])

J = F.jacobian([x, y, z])

print("Stability matrix (Jacobian):")
print(J)
# %% d)
sum_lyapunov_exponents = trace(J)
print("Sum of Lyapunov exponents (trace of Jacobian):")
print(sum_lyapunov_exponents)
```