Lorenz Equations

Kaylin Shanahan 2023

This is a python notebook which uses scipy.integrate.solve_ivp to model the development of a system of Lorenz equations - three linked first order differential equations developed to model the atmosphere. The programme will output a model of the system over a time period of 10 seconds using $10,000$ time steps and plot a graph of z versus x.

The state of the system is described by a point (x, y, z) in a three-dimensional phase-space.

The Lorenz equations that will be used in this programme are:

$$\frac{dx}{dt} = s(y - x)$$

$$\frac{dy}{dt} = rx - y - xz$$

$$\frac{dz}{dt} = xy - bz$$

Where parameter values are $r = 100$, $s = 10$ and $b = 3$ and the initial system state $(x, y, z) = (0, 10, 100)$.

It must be noted that the system is non-linear; tiny perturbations in the initial state of the system will cause to develop along radically different paths.

- Input initial conditions and parameter values
- Input Lorenz equations
- Calculate Lorenz equation solutions
- Ouput plot of the Lorenz System

In [31]:

```
# Use SciPy ODE solver for a second order ODE - Lorenz Equations

# NumPy is needed for our calculations
```

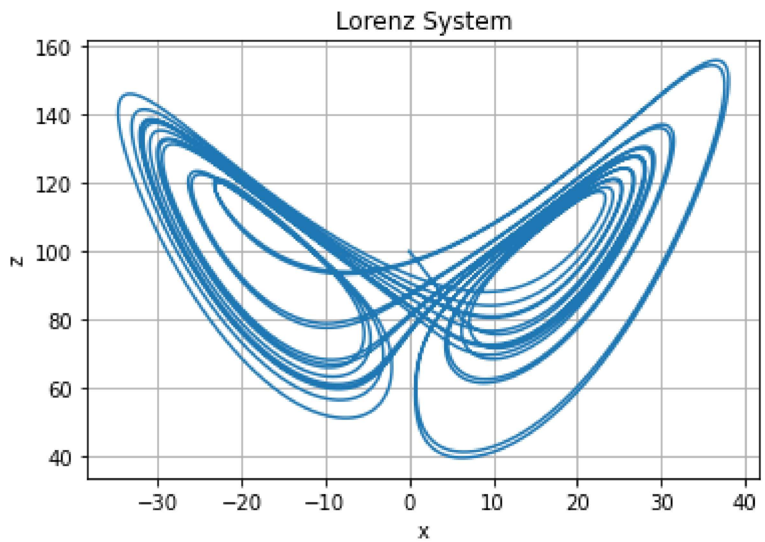
```python
import numpy as np
# SciPy ODE solver is needed for initial value problem
from scipy.integrate import solve_ivp

# Define the Lorenz equations
def lorenz(t, xyz, r, s, b):
    x, y, z = xyz                                    # System state is described by point (x, y, z)
    dx_dt = s*(y-x)                                  # Rate of change of x with respect to time
    dy_dt = r*x - y - x*z                            # Rate of change of y with respect to time
    dz_dt = x*y - b*z                                # Rate of change of z with respect to time
    return [dx_dt, dy_dt, dz_dt]                     # Return array with rates of change
```

In [32]:

```python
# Set system parameter values
r = 100
s = 10
b = 3

# Set initial conditions
xyz_0 = np.array([0, 10, 100])                       # array containing intial system state values
t_0 = 0                                              # Start time
t_max = 10                                           # End time
t_span = [0, 10]                                     # Set time span using array containing start and end time
t_steps = 10000                                      # Set number of time steps
t = np.linspace(t_span[0], t_span[1], t_steps)   # Generate time points

# Call the ODE solver
sol = solve_ivp(lorenz, t_span, xyz_0, args=(r, s, b), t_eval=t)

# Map the solution function
x, y, z = sol.y
```

In [33]:

```python
# Matplotlib is need for plotting
import matplotlib.pyplot as plt
%matplotlib inline

# Plot the Lorenz system
plt.plot(x, z)                   # Plot x versus z
plt.xlabel('x')                  # Label x-axis
plt.ylabel('z')                  # Label y-axis
plt.title('Lorenz System')   # Title of the system
plt.grid()
plt.show()
```



Lorenz System
160
140
120
100
80
60
40
z
−30
−20
−10
0
10
20
30
40
x