Lorenz equations: Jessica Murphy

21/03/2023

Homework 20

This Python notebook demonstrates how to use **scipy.integrate.solve_ivp** to model the Lorenz system , a set of **linked** non-linear ordinary differential equations originally developed to model the atmosphere. Tiny perturbations will cause the system to develop along radically different paths. The state of the system is described by a point (x,y,z) in a three dimensional phase-space.

The Lorenz model is a system of three ordinary differential equations (ODEs) that have chaotic solutions with certain parameter values **s, b, r**, and initial conditions - a point in time of the form **(x,y,z)**. The equations are linked because the x ODE depends on both x and y, the y ODE depends on both x, y and z, and the z ODE depends on both x, y, and z. This program plots the Lorenz system as a parametric function of time. The system of equations are:

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

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

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

A python function is written which takes t and y as parameters, as it is a parametric function of time and SciPy uses an array y for dependant variables. The dependant variables are x,y,z, so these are relabelled as y[0], y[1], y[2]. Then the conditions of the programme are set with s, b and r as parameters where s = 10, b = 3, r = 100, the initial system of the state is defined **(x,y,z) = (0,10,100)**, and the time span is set.

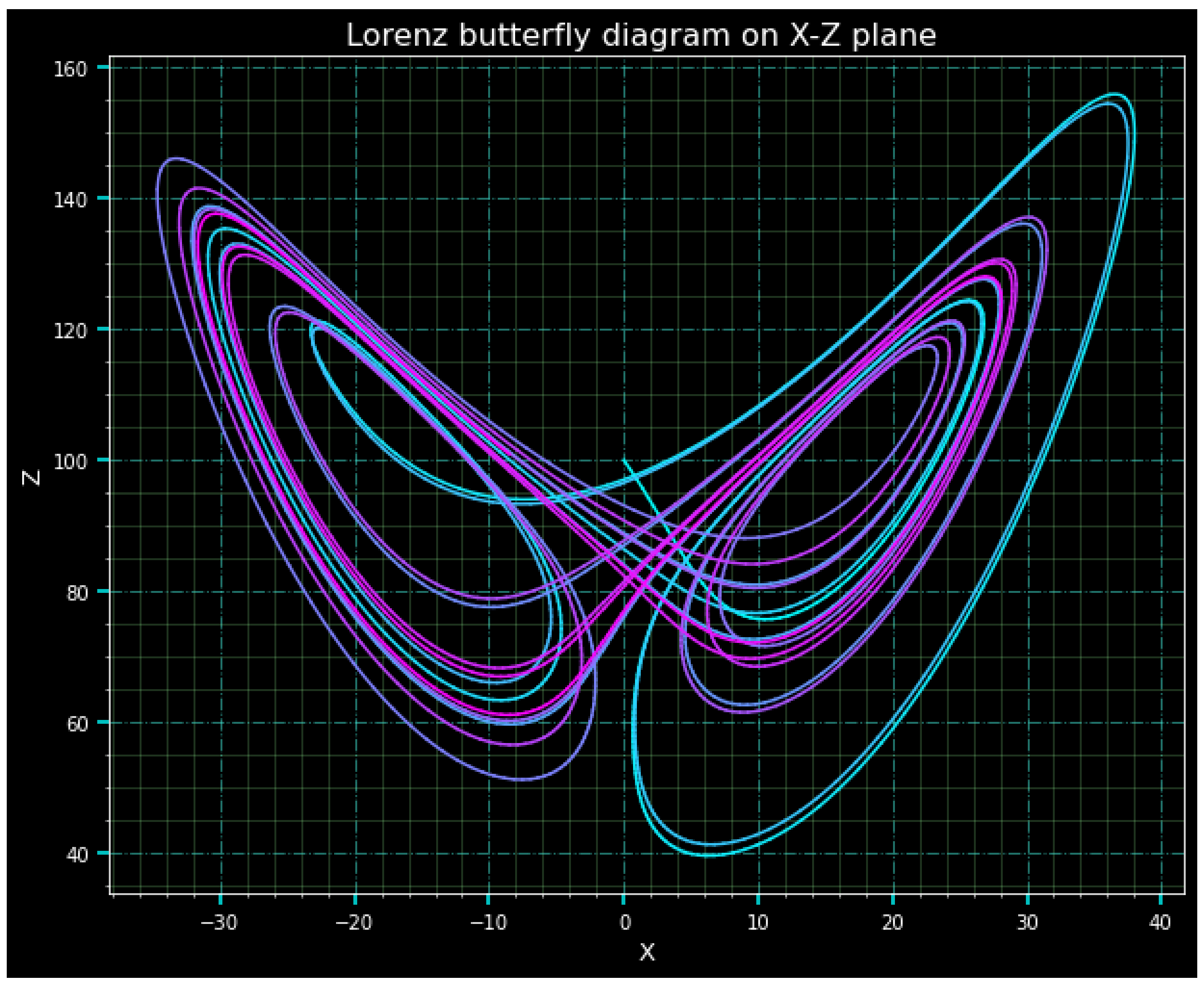
The solution is then computed using the SciPy function 'solve_ivp' which numerically solves a system of ordinary differential equations with the initial values given, using the Runge-Kutta 4th and 5th order method.

Finally, the notebook plots a butterfly diagram of the resulting solution on the X-Z plane using Matplotlib. The resulting plot shows the characteristic butterfly shape of the Lorenz system, demonstrating how tiny perturbations in the initial state of the system can cause drastically different outcomes.

1. Import python library NumPy and from the scipy.integrate sub-package, import the function solve_ivp
2. Define a function to return an array of the ODEs for x,y and z.
3. Set the initial conditions of the parameters s,b,r.
4. Define the initial system state of the point (x,y,z) and assign a value Y_0 for an array containing the initial values.
5. Model the system over the next 10 seconds
   - Define the start & end times and assign an array for this time span
   - Define a linearly spaced time array which is used to evaluate the solution at equal intervals
6. Call the ODE solver solve_ivp
   - Use the Runge-Kutta 4th & 5th method
   - extract the dependant variables x,y,z using the python variable 'soln'
7. Plot the Lorenz butterfly diagram

In [1]:

```python
# Use the SciPy ODE solver for an inital value problem involving the Lorenz equations modelling the development of a system
# the equations are dx/dt = s(y - x), dy/dt = rx-y-xz, dz/dt = xy - bz
# initial conditions; s = 10, b = 3, r = 100

import numpy as np # import python library NumPy
from scipy.integrate import solve_ivp # from the scipy.integrate sub-package use the scipy function solve_ivp for solving ODE initial value problems

# define the differential equations, as a function, with variables y and t
# reassign the x,y,z values to y[0], y[1], y[2] as solve_ivp uses a y array for dependant variables
def dy_dt(t, y):
    dy_dt_0 = s * (y[1] - y[0]) # rate of change of x
    dy_dt_1 = y[0] * (r - y[2]) - y[1] # rate of change of y
    dy_dt_2 = y[0] * y[1] - b * y[2] # rate of change of z
    return [dy_dt_0, dy_dt_1, dy_dt_2] # return an array with all rates of change

# set the initial conditions of the parameters
s = 10
b = 3
r = 100

# define initial system state (x,y,z)
x_0 = 0
y_0 = 10
z_0 = 100
Y_0 = np.array([x_0, y_0, z_0]) # array involving initial system state, allows solve_ivp to deal with a number of linked diff. eqns.

# assigning time values
t_0 = 0 # start time
t_max = 10 # end time - iterate up to a maximum of t_max in seconds
n = 10000 # the number of time steps
t_span = [t_0, t_max] # array containing start and end time - time interval to solve the eqns over
t_eval = np.linspace(t_0 , t_max , n) # linearly spaced time array which is used to evaluate the solution

# call the ODE solver, solve_ivp
# Evaluates the following parameters inside the brackets: the ODE, time interval, the array contating the initial values of the function,
# ..Runge Kutta 4th &5th order, time points, relative tolerance, absolute tolerance
soln = solve_ivp(dy_dt, t_span , Y_0, 'RK45', t_eval , rtol=1e-3, atol=1e-6)

# extracting the (dependant) x,y,z components of the solution using the dot notation by assigning python variable soln
x = soln.y[0]
y = soln.y[1]
z = soln.y[2]

# plot the butterfly diagram
import matplotlib.pyplot as plt # import python library matplotlib to plot
%matplotlib inline

# plot the numerical solution
plt.style.use('dark_background') # for background use style sheet from matplotlib
plt.figure(figsize=(10,8)) # change figure size
plt.tick_params(axis='both', direction='out', length=6, width=2, labelcolor='w', colors='c', grid_color='gray', grid_alpha=0.5) # make axes ticks cyan and writing white
plt.grid(visible=True, color='turquoise',linestyle='-.', linewidth=1) # adjust major grid lines to faint turquoise
plt.minorticks_on() # add minor ticks
plt.grid(visible=True, which='minor', color='palegreen', alpha=0.2, ls='-', lw=1) # adjust minor grid lines to faint green

# make the line change colour by plotting it in segments of length s which change in colour across t
# gives an idea of structure in 3d
s = 5
cmap = plt.cm.cool # use winter colourmap in matplotlib

# use a for loop to iterate over x and z values to plot them and display continuously changing colour
for i in range(0,n-s,s):
    plt.plot(x[i:i+s+1], z[i:i+s+1], color=cmap(i/n), alpha=1)

fontsize = 16 # stating font size wanted for title
axessize = 13 # stating font sizes wanted for axes
plt.title('Lorenz butterfly diagram on X-Z plane ', fontdict={'fontsize': fontsize}) # increase title size
plt.xlabel('X', fontdict={'fontsize': axessize}) # increase axes font
plt.ylabel('Z', fontdict={'fontsize': axessize})


plt.show() # show graph
```



As expected, the plot shows the characteristic butterfly shape of the Lorenz system.