PHYS 220: Fall 2016 Final

Professor Justin Dressel

Due: Dec 13th, 2016

Given three real-valued functions of time $x(t), y(t), z(t)$, consider the following coupled first-order ODEs:

$$\dot{x} = -y - z, \qquad \dot{y} = x + ay, \qquad \dot{z} = b + z(x - c), \tag{1}$$

where $a = b = 0.2$ and c is a parameter that we will tune. Note that this system has a single nonlinear term xz. For this final, you will be exploring the consequences of this nonlinearity.

Problem 0. Create a suitable `README.md` file that describes this assignment (5pt), and sign it to verify that all submitted work is your own (5pt). Link it to Travis.ci (5pt) to automate testing with the nosetests framework (5pt). Place your main python code in a properly commented and formatted file `rossler.py` (5pt), and create a supplementary Jupyter notebook `FinalExam.ipynb` (5pt) that imports your python code and presents your results in a clean and clear way. Done properly, the notebook should contain almost no code (only simple function calls), and should professionally present and discuss of the results instead.

Problem 1. (26pt) In `rossler.py`, write a python function `rossler(c, T=500, dt=0.001)` that implements a 4th-order Runge-Kutta integrator for the Eqs. (1). The function should return a `pandas` dataframe of four numpy arrays of float64 values: `pd.Dataframe("t":t, "x":x, "y":y, "z":z)`. The array `t` should store an evenly spaced set of time points starting from 0 with spacing `dt`, until an upper bound T. The arrays `x`, `y`, and `z` should be pre-allocated as arrays of zeros for efficiency. A single for loop should then integrate Eqs. (1) from the initial conditions $x = y = z = 0$ at $t = 0$ over the array of t values and store the results in the corresponding arrays. (Do not reuse code from previous assignments - write the Runge-Kutta algorithm from scratch specified to this problem.) Write suitable test functions. Use `numba` to speed up your code, and document any resulting improvement.

Problem 2. (14pt) In `rossler.py`, write seven plotting functions that use matplotlib to visualize the solution: `plotx(sol)` (x vs. t, 2pt), `ploty(sol)` (y vs. t, 2pt), `plotz(sol)` (z vs. t, 2pt), `plotxy(sol, T0=100)` (y vs. x, 2pt), `plotyz(sol, T0=100)` (z vs. y, 2pt), `plotxz(sol, T0=100)` (z vs. x, 2pt), and `plotxyz(sol, T0=100)` (3D x-y-z plot, 2pt). In each function, assume that `sol` is the dataframe output by `rossler`. The plots vs. t should show the range $t \in [0, T]$, while the 2D and 3D parametric plots should only include the steady-state time range $t \in [T0, T]$ that discards initial transient points. Plots should be labeled clearly, with x and y having fixed range $[-12, 12]$, and z having fixed range $[0, 25]$. Demonstrate all functionality in your notebook using $c = 2$.

Problem 3. (15pt) Explore the onset of chaotic behavior. In your notebook, show a time plot $x(t)$, a 2D parametric plot $y(x)$, and a 3D parametric plot $z(x, y)$ for each of the following c values: 2, 3, 4, 4.15, 4.2, and 5.7. Show other plots as desired. In your notebook, describe in detail what is happening in each case. Explain how these results relate to the logistic map from the midterm.

Problem 4. (15pt) Examine structure of the local maxima. Create a function `findmaxima(x, N=100)` that isolates *local* maxima of a particular solution array `x`, after discarding the first `N` points (to ignore transient behavior), and returns a numpy array of the local maxima: `xmax`. For each value c in the range $[2, 6]$ (with a small mesh spacing 0.001), solve Eqs. (1) and extract the set of maximal points of `x`, then plot each maximal point (c, x) on the same scatter plot of x vs. c. After plotting all such points, you will have a graph of (the multi-valued function of) the asymptotic local maxima of x vs. c. The range of x in the plot should be from $[3, 12]$. Comment on your findings. What happens if you replace x by y or z in this procedure?