PHYS220/MATH220: Fall 2019 Final

Due: December 11th, 2019

Consider the following set of coupled first-order ODEs for three real-valued functions of time $x(t)$, $y(t)$, $z(t)$, with real-valued constants a, b, c, and initial conditions at $t = 0$:

$$\dot{x} = -y - z, \qquad \dot{y} = x + ay, \qquad \dot{z} = b + z(x - c), \qquad x(0) = y(0) = z(0) = 0. \qquad (1)$$

Recall that the over-dot notation is Newton's short-hand for a time derivative, e.g., $\dot{x} \equiv dx/dt$. For our purposes, we will set $a = b = 0.2$ and leave c as a tunable parameter. Note that this system of equations has only a single nonlinear term zx in the $\dot{z}$ equation. This nonlinearity makes the system impossible to solve analytically, but also leads to very interesting behavior that you will explore numerically in this final.

Problem 0. [25pt] Create a suitable `README.md` file that describes this assignment, and sign it to verify that all submitted work is your own (5pt). Link it to Travis.ci (5pt) to automate testing with the `nose` framework using a test file `test_final.py` (5pt). Place your main python code in a properly commented and formatted file `final.py` (5pt), and create a supplementary Jupyter notebook `Final.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, and should professionally present and discuss of the results instead.

Problem 1. [25pt] In `final.py`, write a python function `solve_odes(c, T=500, dt=0.001)` that integrates the Eqs. (1), given the tunable parameter c, the final time T, and the time step dt (5pt). 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 time points in the domain $[0, T]$ with equal spacing `dt`. The arrays `x`, `y`, and `z` should be pre-allocated as arrays of zeros for efficiency (5pt). A single `for` loop should then integrate Eqs. (1) using the 4th-order Runge-Kutta method 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 (10pt). Write at least one test function to check your implementation (5pt). (Recommended: Use `numba` to speed up your code by decorating any helper functions with `@nb.jit`, 10pt Extra credit.)

Problem 2. [20pt] In `final.py`, write seven plotting functions that use `matplotlib` to visualize the solution: three time plots `plotx(sol)` (x vs t, 2pt), `ploty(sol)` (y vs t, 2pt), `plotz(sol)` (z vs t, 2pt), three 2D plots `plotxy(sol, S=100)` (y vs x, 2pt), `plotyz(sol, S=100)` (z vs y, 2pt), `plotxz(sol, S=100)` (z vs x, 2pt), and one 3D plot `plotxyz(sol, S=100)` (z vs x-y, 3pt). In each function, assume that `sol` is the DataFrame output by `solve_odes`. The time plots should show the domain $t \in [0, T]$, while the 2D and 3D plots should include only the steady-state time domain $t \in [S, T]$ that discards transient behavior. (Hint: there are $N = \lfloor S/dt \rfloor$ points to discard.) 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 `Final.ipynb` using $c = 2$ (5pt).

Problem 3. [20pt] Explore the onset of chaotic behavior. In your notebook, show a time plot $x(t)$, a 2D plot $y(x)$, and a 3D plot $z(x, y)$ for each of the following c values: 3, 4, 4.15, 4.2, and 5.7 (3pt each). Show other plots as desired. In your notebook, describe in detail what is happening in each case. Highlight any similarities you find to the logistic map from the midterm (5pt).

Problem 4. (10pt) Examine structure of the local maxima. Create a function `findmaxima(x, S=100)` that isolates *local* maxima of a particular solution array `x`, after discarding the first `S` points (to ignore transient behavior), and returns a numpy array of the local maxima: `xmax`. Create a function `scatter(dc=0.01)` that for each value c in the range $[2, 6]$ with a small mesh spacing `dc` solves Eqs. (1) and extracts the set of maximal points of `x`, then plots 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. (Recommended: Use `@nb.jit` to speed up your code; increase mesh spacing `dc` as needed if runtime or memory are exceeded.)

Problem 5. (Extra credit: 10pt) Create an animated gif on the Schmid cluster `schmidcluster.chapman.edu`. Modify the `scatter` function from problem 4 to add a keyword argument `cmax=6` that sets a tunable maximum c value for the scatter plot, such that the scatter plot is always plotted with horizontal domain $[2, 6]$ and vertical range $[3, 12]$. Also modify the `scatter` function to add a keyword argument `gif=False` that will toggle whether the plots are created to be shown immediately in a notebook (by default), or to be exported to `png` files without being shown to the screen (when `gif=True`). (Hint: Look carefully at HW12 for how to do this.) Create an executable `bash` script `gengif.sh` that exports one plot as `png` per three increments of `dc` in the `scatter` function, by tuning the `cmax` value appropriately. Name each `png` a distinct name like `frame000.png`. After generating all the plots as image files, have the script use the `convert` command to create an animated gif, then optimize that gif as recommended in HW 12. Run this script on the Schmid cluster to create your animated gif, then commit only the final optimized gif to your final repository.